

AD-A042 264

DOTY ASSOCIATES INC ROCKVILLE MD

F/G 9/2

SOFTWARE COST ESTIMATION STUDY. VOLUME I. STUDY RESULTS.(U)

JUN 77 J H HERD, J N POSTAK, W E RUSSELL

F30602-76-C-0182

UNCLASSIFIED

TR-151

RADC-TR-77-220-VOL-1

NL

1 OF 4  
ADA042264



64

AD A 042264

RADC-TR-77-220, Volume I (of two)  
Final Technical Report  
June 1977

SOFTWARE COST ESTIMATION STUDY  
Study Results

Doty Associates, Inc.

Approved for public release; distribution unlimited.

AD No. \_\_\_\_\_  
BDC FILE COPY

ROME AIR DEVELOPMENT CENTER  
Air Force Systems Command  
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED:

*Alan N. Sukert*

ALAN N. SUKERT, Captain, USAF  
Project Engineer

APPROVED:

*Alan R. Barnum*

ALAN R. BARNUM  
Assistant Chief  
Information Sciences Division

ADDITIONAL FOR	White Section	<input type="checkbox"/>
NTIS	Buff Section	<input type="checkbox"/>
DIG		
UNANNOUNCED		
JUSTIFICATION		
BY		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	Avail. and/or Special	
<i>RF</i>		

FOR THE COMMANDER:

*John P. Huss*

JOHN P. HUSS  
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DAP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-77-220 - Volume I (of two)	2. GOVT ACCESSION NO. 1	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE COST ESTIMATION STUDY, Volume I, Study Results	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 23 Feb 76 - 23 Feb 77	6. PERFORMING ORG. REPORT NUMBER Technical Report #151
7. AUTHOR(s) James H. Herd, John N. Postak, William E. Russell, Kenneth R. Stewart	8. CONTRACT OR GRANT NUMBER(s) F30602-76-C-0182	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Doty Associates, Inc. 416 Hungerford Drive Rockville MD 20850	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55811404	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441	12. REPORT DATE June 1977	13. NUMBER OF PAGES 202
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	15. SECURITY CLASS. (of this report) UNCLASSIFIED	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Captain Alan N. Sukert (ISIS)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Cost Estimates      Software Development Costs Business Software Costs      Software Development Time Command and Control Software Costs      Software Size Cost Analysis      Utility/Support Software Costs Scientific Program Software Costs		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The study identified factors that have an adverse effect on software cost estimates, determined their impact on software cost estimates, discussed methods for controlling the effect of these factors, and developed an overall methodology for estimating the costs of software development. In addition to a generalized model for estimating software development costs, separate models have been generated for estimating the development cost of command and control, scientific, utility, and business software.		

DD FORM 1 JAN 73 1473

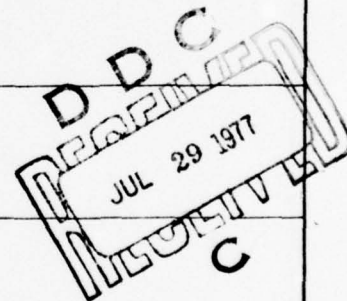
EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

406 593

mt



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## PREFACE

This final report entitled "Software Cost Estimation Study, Volume I: Study Results," was prepared by Doty Associates, Inc. (DAI), for the Information Systems Division (ISIS), Rome Air Development Center (RADC), Griffiss Air Force Base, New York 13441. Inclusive dates of research were from 23 February 1976 through 22 February 1977. Submittal date was 23 March 1977. This technical effort was accomplished under Contract number F30602-76-C-0182. The Air Force Project Manager for this project was Captain Alan N. Sukert, USAF of RADC.

# CONTENTS

		<u>Page</u>
Paragraph 1	INTRODUCTION . . . . .	1
1.1	Background . . . . .	1
1.2	Study objectives . . . . .	3
1.3	Summary of results . . . . .	6
1.4	Organization of the report . . . . .	6
2	STUDY APPROACH . . . . .	8
2.1	Task 1. Identification of Factors Affecting Estimates . . . . .	8
2.1.1	Data collection . . . . .	9
2.1.2	Data analysis . . . . .	11
2.2	Task 2. Examination of Techniques for Con- trolling Effects . . . . .	11
2.3	Task 3. Development of an Approach for Software Cost Estimation . . . . .	12
3	FACTORS AFFECTING THE RELIABILITY OF SOFTWARE SIZE AND COST ESTIMATIONS . . . . .	14
3.1	The data . . . . .	14
3.2	The analysis . . . . .	17
3.3	The results . . . . .	21
3.3.1	Purchaser domain factors . . . . .	21
3.3.2	Developer domain factors--preparation procedures . . . . .	53
3.3.3	Developer domain--project management procedures . . . . .	68
3.3.4	Developer domain--determining actual costs . . . . .	76
3.3.5	Other factors--type of application . . . . .	80
4	TECHNIQUES FOR IMPROVING RELIABILITY OF SOFTWARE COST ESTIMATION . . . . .	84
4.1	Criteria for selection of techniques . . . . .	84
4.2	Employment of techniques . . . . .	84
4.3	Control of the primary factors . . . . .	86
4.3.1	Special displays . . . . .	87
4.3.2	Data management . . . . .	87
4.3.3	Definition of operational requirements . . . . .	88
4.3.4	Changes in operational requirements . . . . .	88
4.3.5	Interface to design . . . . .	89
4.3.6	Response time requirements . . . . .	90
4.3.7	Time (hardware) constraint . . . . .	91
4.3.8	Memory (hardware) constraint . . . . .	92
4.3.9	Time and memory (hardware) constraint . . . . .	93
4.3.10	First development on computer . . . . .	93
4.3.11	Concurrent development of hardware . . . . .	93
4.3.12	Time CPU specified in schedule (hardware constraint) . . . . .	94
4.3.13	Requirement for innovation . . . . .	94
4.3.14	Language requirements . . . . .	94

	<u>Page</u>
Paragraph 4.3.15	Quality requirements . . . . . 95
4.3.16	Reliability requirements . . . . . 98
4.3.17	Testing requirements (including verification and validation) . . . . . 99
4.3.18	Transportability requirements . . . . . 100
4.3.19	Maintenance requirements . . . . . 101
4.3.20	Development schedule . . . . . 101
4.3.21	Communications . . . . . 101
4.3.22	Developer using another activity's computer . . . 102
4.3.23	Programmer access to computer . . . . . 102
4.3.24	Operational site development . . . . . 103
4.3.25	Development and target computer different . . . 103
4.3.26	Number of development locations . . . . . 104
4.3.27	Programming environment . . . . . 105
4.3.28	Support software . . . . . 106
4.3.29	Programming facilities . . . . . 106
4.3.30	Multiple software utilization sites . . . . . 107
4.3.31	End user requirements . . . . . 107
4.3.32	Design complexity . . . . . 107
4.3.33	Design stability . . . . . 108
4.3.34	Modern programming techniques . . . . . 108
4.3.35	Sizing estimate error . . . . . 109
4.3.36	Definition of instruction . . . . . 110
4.3.37	Personnel mix by type . . . . . 110
4.3.38	Work Breakdown Structure (WBS) . . . . . 111
4.3.39	Cost/Schedule Control Systems Criteria (C/SCSC) . . . . . 111
4.3.40	Amount and method of cost data collection . . . 112
4.3.41	Secondary resources . . . . . 113
4.3.42	Software applications . . . . . 113
5	METHODOLOGY FOR ESTIMATING SOFTWARE
	DEVELOPMENT COSTS . . . . . 116
5.1	Development of the estimating algorithms . . . 117
5.1.1	Time required for development . . . . . 117
5.1.2	Cost estimating algorithms . . . . . 118
5.1.3	Software sizing . . . . . 170
5.1.4	Secondary resources . . . . . 175
5.2	Proposed methodology and existing directives . . 175
5.3	Study results . . . . . 176
5.4	Recommendations . . . . . 183
Appendix A - Quantitative Data Bases Obtained from literature . . . A-1	
Bibliography . . . . . Biblio-1	

# FIGURES

	<u>Page</u>
FIGURE 1. Source code data for total population. . . . .	19
2. Source code data for total population (I<10,000) . . . . .	20
3. Effect of time and memory constraints on software productivity. . . . .	32
4. Software reliability economic considerations .	40
5. Software development time estimator. . . . .	44
6. Estimated distribution of resources for a medium-large project (~100,000 object instructions). . . . .	46
7. Hardware/Software cost trends. . . . .	69
8. Anticipated shape of estimator for optimal duration of software development . . . . .	119
9. Baseline relationships for software programs .	124
10. Relationship between program size in source code and development manpower for total population . . . . .	125
11. Relationship between program size in source code and development manpower for total population (I<10,000). . . . .	126
12. Relationship between program size in object code and development manpower for total population . . . . .	127
13. Relationship between program size in object code and development manpower for total population (I<10,000). . . . .	128
14. Comparison of actual and estimated development manpower for total population. . . . .	130
15. Manpower estimation model for software . . . . .	131
16. Command and control program source code data .	132
17. Command and control program source code data (I<10,000) . . . . .	133
18. Baseline manpower relationships for command and control programs . . . . .	135
19. Relationship between program size in source code and development manpower for command and control programs . . . . .	136
20. Relationship between program size in source code and development manpower for command and control programs (I<10,000). . . . .	137
21. Relationship between program size in object code and development manpower for command and control programs . . . . .	138

# FIGURES

	<u>Page</u>
FIGURE 22. Relationship between program size in object code and development manpower for command and control programs ( $I < 10,000$ ) . . . . .	139
23. Comparison of actual and estimated development manpower for command and control software programs . . . . .	140
24. Manpower estimation model for command and control software . . . . .	141
25. Comparison of actual and estimated development manpower for command and control software programs (multivariate model) . . . . .	142
26. Scientific program source code data . . . . .	143
27. Scientific program source code data ( $I < 10,000$ ) . . . . .	144
28. Baseline manpower relationships for scientific programs . . . . .	145
29. Relationship between program in source code and manpower for scientific programs . . . . .	147
30. Relationship between program in source code and manpower for scientific programs ( $I < 10,000$ ) . . . . .	148
31. Relationship between program in object code and development manpower for scientific programs . . . . .	149
32. Relationship between program size in object code and development manpower for scientific programs ( $I < 10,000$ ) . . . . .	150
33. Comparison of actual and estimated development manpower for scientific software programs . . . . .	151
34. Manpower estimation model for scientific software . . . . .	152
35. Business program source code data . . . . .	153
36. Business program source code data ( $I < 10,000$ ) . . . . .	154
37. Baseline manpower relationships for business programs . . . . .	156
38. Relationship between program size in source code and manpower for business programs . . . . .	157
39. Relationship between program size in source code and manpower for business programs ( $I < 10,000$ ) . . . . .	158
40. Relationship between program size in object code and development manpower for business programs . . . . .	159

# FIGURES

	<u>Page</u>
FIGURE 41. Relationship between program size in object code and development manpower for business programs (I<10,000) . . . . .	160
42. Comparison of actual and estimated development manpower for business software programs . . .	161
43. Manpower estimation model for business software . . . . .	162
44. Utility program source code data. . . . .	163
45. Baseline manpower relationships for utility programs . . . . .	164
46. Relationship between program size in source code and development manpower for utility programs . . . . .	166
47. Relationship between program size in object code and development manpower for utility programs . . . . .	167
48. Comparison of actual and estimated development manpower for utility software programs. . . .	168
49. Manpower estimation model for utility software.	169
50. Suggested utilization of estimating relation- ships for development manpower . . . . .	181

# TABLES

		<u>Page</u>
TABLE 1	FACTORS ANALYZED FOR EFFECTS . . . . .	15
2	PURCHASER DOMAIN . . . . .	22
3	DEVELOPER DOMAIN PREPARATION PROCEDURES . . . . .	54
4	ACCURACY OF SIZING PARAMETER . . . . .	61
5	DEVELOPER DOMAIN PROJECT MANAGEMENT PROCEDURES . . . . .	70
6	DEVELOPER DOMAIN DETERMINING ACTUAL COSTS . . . . .	78
7	TYPE OF APPLICATION . . . . .	81
8	FACTORS AFFECTING COST ESTIMATION ACCURACY FOR WHICH CONTROLS ARE PROPOSED . . . . .	85
9	STRUCTURE OF THE DATA SAMPLE . . . . .	122
10	IMPACT OF FACTORS ON SOFTWARE COST ESTIMATION . . . . .	178

## EVALUATION

The increased importance of software for military applications, coupled with the increased expenditures by both the military and civilian communities for the development of software, has brought about an increased awareness of the present high cost of software and the consistent inability to accurately predict the cost of software projects. This need for producing lower cost software and for more accurately estimating software costs has been expressed in such documents as the Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (November 1975) and the Summary Notes of a Government/Industry Software Sizing and Costing Workshop (November 1974) (ESD-TR-76-166), as well as in numerous Government and industry sponsored symposia. As a result, several efforts have been initiated to develop better methods for estimating software costs. However, these efforts have not adequately considered the basic underlying factors that affect software sizing and cost estimates, and have not, in most cases, considered non-linear software cost estimating relationships.

This effort was initiated in response to the need to better understand and control those factors that adversely affect software sizing and cost estimates, and fits into the goals of RADC TPO No. 5, Software Cost Reduction (formerly RADC TPO No. 11, Software Sciences Technology), in particular the area of Software Quality (Modeling). The report concentrates on the identification of over forty factors that are shown to have an adverse impact on the accuracy of software sizing and cost estimates, and the formulation of methods for minimizing the effects of these adverse factors, in both the software developer and purchaser domains. The importance of being able to identify and minimize these adverse factors is that it will enable software cost analysts, as well as software managers, to more accurately predict the costs of software projects, by recognizing those factors that have to be considered when making software cost estimates during the various phases of the software development cycle. This, in turn, will enable software managers to better control the costs of software projects and thus greatly reduce the potential for severe cost overruns that presently exists. Finally, the overall methodology proposed in this report will provide methods that future software cost estimators can use to obtain accurate cost estimates during each phase of a software development project, which will greatly aid in the preparation of independent software cost estimates for use in project evaluation.

*Alan N. Sukert*

ALAN N. SUKERT, Captain, USAF  
Project Engineer

## 1. INTRODUCTION

### 1.1 Background

Since the advent of modern computers, it has been common for the cost and time required to develop software, particularly for large programs, to exceed initial estimates. In addition, the increased sophistication of software applications over the past ten years has made these erroneous estimates more significant in terms of absolute costs and the percent impact on total system cost. The erroneous estimates can be caused by any one or a combination of numerous factors. Among the most critical factors are changes in the operational requirements, which affect the functional specifications of the software. However,, even when the specifications have been fixed, it has been difficult to project the resource requirements accurately. The primary resource--manpower--varies widely in productivity and quality and is affected in a complex manner by the multi-dimensional environment in which the software is developed. Secondary resources such as machine time and publications support are frequently unavailable at appropriate times. In addition, information with which to develop estimates of resource requirements, such as program size, program language, and type computer are not always available on a timely basis. And, if these items are defined, the system can be aggregates of so many elements, organizational interactions, logistical considerations, etc., that it is difficult to assess the scope of the work accurately.

Essentially, the field of software management and engineering is still in its infancy, especially as it relates to deriving cost estimates of software development. The field has evolved to the state where the cost of a software package is generally developed by estimating the number of instructions to be delivered with the package (i.e., size), and multiplying the size by a cost factor based on average personnel

productivity. The Air Force, other DoD and government agencies, and commercial organizations have found this method to be inadequate. This simplistic approach has resulted in large cost overruns in several software development projects. Size estimates have been observed to be erroneous in many cases by a factor exceeding 3, and it is common to have a productivity factor that has a standard deviation 2.5 times the expected value. With such large variances associated with the two factors most commonly used in software cost estimation, it is not surprising that large software cost overruns occur. Of the two factors, size is the most important since a misestimation in this parameter can have an impact on hardware as well as software costs.

With a long history of cost overruns, program managers often wonder why software development costs cannot be predicted as accurately as the costs of engineering tasks. During the past several years, extensive work has been performed in the development of improved techniques and guides for the prediction of software costs. However, the procedures or models evolving from these studies have unfortunately been demonstrated to be inaccurate estimators--usually for the reasons discussed previously--erroneous estimation of the size of the software packages and/or of programmer productivity. Major software development contractors have been sensitive to the need for improved techniques of cost estimation because of pressures exerted by customers who have experienced or are projecting serious cost overruns. As of late, the techniques offered by the developers usually have involved improved "software management" which has had a relatively minor impact on the resultant discrepancies between estimated and actual costs. Various proposed control schemes and procedures for delineating work packages have had marginal success. There is evidence that failure of the control mechanisms is due to improper implementation, unresponsive management, inaccurate interpretation and inadequate analysis of reported data, and untimely reporting of problems.

Unfortunately, insufficient emphasis has been given to the analysis of cause and effect relationships of software development. Although

there has been extensive discussion and some evidence presented in the literature as to the impact of selected practices of the developer and purchaser on software costs, comparatively little analysis had been performed to assess quantitatively a broad spectrum of these effects and to isolate controls for mitigating these effects.

### 1.2 Study objectives

In response to this need, Doty Associates, Inc. (DAI), began a study in February 1976 of "Software Cost Estimation," under Contract Number F30602-76-C-0182 with the Information Sciences Division of the Rome Air Development Center (RADC/IS). An overall objective of the study was to reduce the variance between estimated and actual costs of software development. This objective was to be effected through a detailed understanding of the factors that influence software size and cost estimation, the development of techniques for improving the reliability of these estimates, and a recommended methodology for estimating the cost of software development. The research was structured into three tasks, specifically to:

- Identify those factors that cause unreliable software cost estimates, including deficient practices of the software developer and the software purchaser, that lead to inaccurate software cost estimation.
- Examine techniques for eliminating or mitigating the effects of those factors identified as having adverse impacts on software cost estimation. Techniques determined to be of value should be easily implemented in a timely manner and should address problems related to both the role of the purchaser and the developer in software cost estimation.
- Develop an overall approach to software cost estimation which integrates all aspects of the problem concerning the developer and the purchaser. The approach should recommend standardized procedures for ensuring that future software cost estimations are reliable.

An extensive literature search was undertaken to assess the methods and results of related studies performed previously, and to accumulate data to supplement that in DAI files. Correlation and regression techniques were used to identify and evaluate those factors having an impact on software costs. Non-linear regression and linear multivariate regression techniques were used to derive models for estimating the time and manpower required to develop software. Then, the impacts of the factors on the estimation process were analyzed quantitatively; the analysis results permit adjustments to the estimates which reflect characteristics of the software, constraints to the development, and parameters of the development environment. Not all the effects could be quantified; however, values were derived for factors having the most significant effect on costs. Proposed controls to alleviate adverse effects of the factors, the models and the adjustment ratios have been integrated into a methodology for improved estimations of software development costs. The approach, discussed herein, has been incorporated into a guide (Volume II of this report) for improved software cost estimation for use by government and contractor personnel.

One of the serious deficiencies in the available historical data was that it frequently did not include estimated and actual resources expended on software development programs. This data was considered important to isolating the factors that affect the accuracy of an estimate. Only data received from the Air Force Data Systems Design Center (AFDSDC) contained this important information; but even this data proved inadequate for use in this study. It was initially intended that this type of information be obtained from Air Force SPOs involved in software development; however, the data could not be made available. This adversely constrained the quality of the study results.

The collection and analysis of the data revealed that there are two other important deficiencies inhibiting the establishment of reliable

software cost estimation procedures: (1) precise and rigorous definitions of the data elements collected, and (2) a systematic method of collecting such data. This revelation is not new, for it has been discussed in numerous prior studies by various authors. Nevertheless, because of the importance of accurate data to the results of this and future studies, and the intended use of the reported cost estimation methodology, it is considered imperative that the concern be expressed here.

Without specific definitions of data elements, it might be assumed that data collected at various activities and in various reports and studies are consistent and therefore can be aggregated. In practice, due to the inadequacies and inconsistency of definitions, supposedly similar data is often quite different. (An important example of this is the various interpretations of what actually constitutes a source instruction count for the purposes of determining the size of an actual or proposed software program.) Thus, resources should be devoted to the collection of data in a rigorous manner so that specifically defined data is collected. In addition, data collection and analysis personnel should be fully knowledgeable of the nuances in the data definitions to ensure that proper data is being collected and utilized. More accurate and consistent reporting of data would have improved the results of this study significantly.

If the collection process was automated, the data would be more precise and consistent. This could also possibly lower the cost of collecting the data, although an initial investment cost would have to be absorbed to implement the automated procedures. (As an example, some extension of the procedures outlined for Programming Support Libraries in the RADC Structured Programming Series, [85], could be implemented for the automated collection of sizing data.)

### 1.3 Summary of results

Through the literature review, encompassing 137 documents, and interviews with several software development agencies, over 100 factors were identified as potentially affecting software costs. The effects of 42 of these factors were found to be significant; however, data limitations only permitted the effects of 29 to be quantified.

Control procedures have been proposed for mitigating the effects of the 42 factors. These controls, and algorithms for estimating the cost of software development, have been integrated into a methodology for improved software cost estimation. As part of this effort, models were derived for estimating the manpower requirements of software development by software applications; for estimating the time required for software development; and for estimating the size of software by application. The resultant methodology has been structured as a formalized guide for improved estimation of software development costs to be used by software development managers and software developers to assist in planning, budgeting, control, and development functions, as appropriate. The guide is presented as a separate volume of this report.

### 1.4 Organization of the report

The final technical report of the software cost estimation study consists of two volumes. Volume I contains the analytical study results, and Volume II is a management guide presenting a time phased overall methodology for estimating software development costs. This volume consists of the following:

- Section 1 contains the introduction to the report, with comments on software cost estimating, and a description of the objectives and tasks of this study effort.
- Section 2 discusses the study approach and methodology, the literature search, and the data collection and analysis.

- Section 3 discusses the factors affecting software cost estimating, including those in the developer and purchaser domains.
- Section 4 presents proposed techniques for improving the reliability of software cost estimates from the factors identified in Section 3 as having an adverse impact.
- Section 5 contains study results and recommendations which encompass the overall methodology for software cost estimating.
- Bibliography presents the technical literature reviewed in this study effort.
- Appendix A describes quantitative data bases obtained from the literature.

## 2. STUDY APPROACH

The study encompassed the following functions:

- a literature search to evaluate the results of similar study efforts,
- the collection of data to form a base with which to draw conclusions relative to software cost estimation,
- the analysis of data to identify the factors having adverse effects on software cost estimation,
- the selection of procedures for controlling the effects, and
- the development of a method for estimating software development costs.

Since it was intended that the cost estimating method include an analytical model, emphasis was given to the collection of quantitative historical data which define the characteristics of software programs developed previously, the resources expended in developing the software, and the environment in which the software was developed.

### 2.1 Task 1. Identification of Factors Affecting Estimates

The objective of this task was to identify factors that cause unreliable estimates of software cost. The factors were partitioned into three categories: (1) those primarily the result of software purchaser practices, (2) those primarily the result of software developer practices, and (3) those caused by other influences. This effort consisted primarily of two parts; the first was the collection of historical data relative to software development programs; the second was the analysis of the data to isolate those factors having an adverse impact on software costs. In the purchaser domain, the areas examined included performance specification, task identification, program management, and inherent uncertainties in software cost estimation. The developer domain was

structured into three sub-domains: preparation procedures, project management procedures, and methods of determining actual costs. The third domain involved software application.

2.1.1 Data collection. The data collection consisted of a review of DAI corporate files for applicable data, a literature search, and personal interviews relative to software development and associated estimated and actual costs. Besides the data existing in the corporate files, the most significant quantitative data was obtained from the literature. The Bibliography reflects the reports reviewed; however, only a small number of the documents contained data suitable for analysis. The data bases from which relevant information were derived are discussed in Appendix A.

Interviews were conducted with management and staff personnel at several commands, agencies and companies to obtain applicable data, to discuss the experience of the individuals (or groups) in estimating software development costs, and to identify other potential sources of data. Among the agencies interviewed were the following:

- AF Aeronautical Systems Division, Wright-Patterson AFB, Ohio
- AF Data Systems Design Center, Gunter AFS, Alabama
- AF Electronic Systems Division, Hanscom AFB, Massachusetts
- HQ AF Logistics Command, Wright-Patterson AFB, Ohio
- AF Space and Missile Systems Organization, Los Angeles AFS, California
- Army Ballistic Missile Defense, Huntsville, Alabama
- Army Missile Command, Huntsville, Alabama
- DoD Computer Institute, Washington, D.C.
- General Research Corporation, Santa Barbara, California
- IBM Corporation, Westlake, California
- Johns Hopkins Applied Physics Laboratory, Columbia, Maryland
- Mitre Corporation, Washington, D.C., and Bedford, Massachusetts
- Naval Electronic Systems Command, Washington, D.C.
- Office of the Secretary of Defense, Washington, D.C.

These were accomplished through personal visits and by telephone. For the most part, the interviews only provided qualitative data and literature references. Numerous other contacts were made by telephone and correspondence.

From the commencement of the study effort, it was recognized that there would be certain problems associated with the collection of data to support the analysis. Among these were the acknowledged difficulty in recouping historical data, especially detailed quantitative data (as opposed to qualitative or judgmental), the restructuring of the dynamics of software development programs which encompassed changing specifications, and outside influences not always documented or remembered. The dynamics of the software development process make it vital that data be obtained relative to the periods before, during, and after development of the software. A pre-development cost estimate might have been based on specifications vastly different from those of the final product. Thus, the unreliability of the cost estimate may not have been entirely the fault of the cost estimation technique, but of a poorly written specification of the product delivered, or of specification revisions resulting from changing operational requirements.

The difference between the initial cost estimate and the final actual cost, therefore, was considered very important data. However, there is little, if any, data of this type available in the literature. It was included in the AF Data System Design Center (AFSDC) Planning and Resource Management Information System (PARMIS) data. As noted previously, the data was not analyzed because of deficiencies. It was intended that this type of information be obtained from Air Force SPOs through in-depth interviews and reviews of program data. However, the data could not be made available.

Another major problem, which became evident in the data collection and had a significant effect on the data analysis, was the lack of consistency in the definitions associated with software and its development. Where possible, DAI evaluated the consistency and adequacy of definition before the data was analyzed. In some cases, data was deleted because it was nebulous or erroneous.

2.1.2 Data analysis. Because of inconsistencies in the data, it was not possible to integrate the ten available data bases outlined in Appendix A. Therefore, each was analyzed independently to determine if factors affecting costs could be identified. Since none of the data samples included both initial estimated and final actual costs, the factors which affected the accuracy of the estimates could not be identified explicitly. Consequently, implicit identification was accomplished by evaluating factors considered determinants of software costs and by presuming that errors in estimating the magnitude and/or effect of these factors were most responsible for inaccurate cost estimates. Regression techniques were used to evaluate the expected cost impact of the factors. The effects were assessed for consistency among data sets and were compared with information presented in the literature.

For those factors which could not be measured quantitatively, or were unsuitable for quantitative evaluation based on the data available, judgmental statements were made as to their impact based on the perspective gained from the overall analysis, prior corporate experience, and the literature search.

## 2.2 Task 2. Examination of Techniques for Controlling Effects

Techniques were explored for controlling the factors identified as having an adverse impact on the reliability of software cost estimation. This task essentially involved the hypotheses of control mechanisms and the analysis of data to determine if the hypotheses were substantiated. In addition, techniques that have been tried and are currently in use for controlling the impact of these factors were examined for their effect on the reliability of the cost estimate in either quantitative or qualitative terms. Procedures have been presented for controlling the effects of each of the factors found to have significant impact on software development costs.

### 2.3 Task 3. Development of an Approach for Software Cost Estimation

The objective of this task was to develop a methodology for generating improved estimates of software development costs. A set of mean value algorithms for resource expenditures was developed in which size (number of object or source instructions) was the only known parameter. A procedure to adjust these estimates was devised, using the quantitative impacts identified in Task 1 modified by the controls suggested in Task 2. The resultant procedure is intended to be dynamic in nature, since different algorithms are to be selected depending on where the project is situated in the software conceptual and development phases. The accuracy of the estimates should improve progressively as the project proceeds through the development cycle and more definitized data are made available upon which to base the software cost estimates.

Recommendations are presented as to when controls should be implemented throughout the development cycle of the software. Those important to the conceptual and analysis phases are relevant to requirements analysis, software specifications, task identification and definition, tradeoff studies, Work Breakdown Structure (WBS) development, and cost and sizing procedures. Controls effecting the software development phase include status and cost reporting, on-site monitoring, and program management.

Individual models have been developed for estimating the man-months of effort required for developing command and control, scientific, utility, and business type software programs, and for all software programs. For some categories of programs, models have been developed for programs of less than, and equal to or greater than 10,000 instructions (both object and source code). Then, the procedure indicates how the factors affect the algorithms in terms of the increase or decrease in productivity anticipated. In addition, the approach includes an estimation for the duration required for developing software and presents procedures for estimating the size of programs.

These methods have been incorporated into a formalized procedure for improving the reliability of estimates for software development costs. The approach, presented in Volume II of this report, is a guide for estimating the resource requirements of software development. It also provides information with which to evaluate the impact of selected factors on these resource requirements, and mechanisms to mitigate the adverse effects on these factors. The approach proposes standards and procedures to which the purchaser should adhere in the preparation of the Statement of Work (SOW) for the Request for Proposal (RFP). They impose controls (constraints) on prospective developer responses. Further, parameters and procedures are recommended for cost monitoring and control by the purchaser to ensure that actual costs remain within acceptable tolerances of the cost estimate.

Implementation of the overall approach will impose certain constraints upon the software purchaser and the developer that may initially appear to restrict the dynamics of the development. However, the constraints must be considered in tradeoffs among the following: the need to keep actual costs within tolerances of the cost estimates, the need to acknowledge the changing requirements in a viable command environment, and the desirability of modifying initial software development specifications to incorporate elements of new technologies. Further, the complexity and size of a specific software development project, and the duration (time frame in which the project must be completed) will, of course, also affect the constraints and tradeoffs. The techniques incorporated in the cost estimating approach are not new; however, the application of the methodology to a time phased management structure for software development is unique. Since Department of Defense and Air Force directives related to software development contain adequate, but sometimes defused, policy guidance, drastic revisions to the directives are not presently recommended. Instead, emphasis is given to the proper use of the directives in conjunction with the proposed methodology.

### 3. FACTORS AFFECTING THE RELIABILITY OF SOFTWARE SIZE AND COST ESTIMATIONS

In Task 1 of the study, DAI identified factors having an impact on the reliability of software size and cost estimations. Limitations of the data prevented an exhaustive analysis of all factors that could have an impact on software development. Nevertheless, the scope of the factors analyzed was comprehensive and it is unlikely that a factor having a significant impact was excluded. The unidentified factors are not expected to affect the accuracy of the resultant cost estimation methodology appreciably.

In this study, the factors were divided into three broad categories or domains: (1) those that are primarily the results of purchaser practices, (2) those that are primarily the result of developer practices, and (3) type of application. The developer domain categories are further divided into sub-domains and all three domain categories are divided into areas. Some factors are applicable to more than one domain but were assigned to the domain in which they were most applicable.

This section discusses the factors investigated, the type of investigation performed (i.e., quantitative, qualitative, or both), and the results of the investigations in terms of their impact on software cost estimations. Table 1 provides a listing of all the factors investigated in this study. It should be noted that in a number of instances where impacts were identified, the magnitude of the effect could not be assessed due to limitations of the data.

#### 3.1 The data

Ten sources of data, described synoptically in Appendix A, were reviewed for their applicability to this study. These included four sets

TABLE 1. FACTORS ANALYZED FOR EFFECTS

Software Characteristics:

- Size
- Complexity
- Operation
  - Application
  - Response time/real time operation
  - Frequency of operation
  - Special displays
  - Number of ADP centers
  - Data base management
  - Transportability
  - Reliability
- Data Base
  - Number of words
  - Number of classes of items
- Messages
  - Number of inputs
  - Number of outputs

Development Characteristics:

- Requirements
  - Development time
  - Quality
  - Testing
  - Changes during development
  - Design interfaces
  - Language
  - Maintenance
  - Need for innovation
  - Stability of design
  - Adequate definition of operation requirements
  - Developer participation in design
  - When CPU is specified during software development
- Hardware
  - Memory constraint
  - CPU time constraint
  - Concurrent development of ADP hardware

TABLE 1. FACTORS ANALYZED FOR EFFECTS (Continued)

- Experience
  - Programmer experience with language
  - Programmer experience with application
  - Customer experience with software development
  - First programming effort on computer
- Environment
  - Average turnaround time of batch processing during development
  - Number of management controls during development
  - Number of agencies that must concur with software design
  - Software developed on computer operated by another agency
  - Availability of computer to programmers
  - Use of time share in software development
  - Software developed at site other than operational site
  - Software developed on computer other than target computer
  - Number of locations at which software is developed
  - Software developed at military organization
  - Effectiveness of communication during development
  - Availability of secondary resources
  - Availability of support software
  - Programming facilities
  - Programming environment
- Management
  - Work breakdown structure
  - Cost/schedule control system
  - Data collection
  - Personnel mix
  - Modern programming techniques
  - Definitions
  - Total pages of documentation
  - Total pages of external documentation
  - Number of users

of data reported by the Systems Development Corporation (SDC), [46, 49, 120, 132], two sets by the Planning Research Corporation (PRC), [13, 104], two sets by the International Business Machines Corporation (IBM), [86, unpublished], one by Logicon, Incorporated, [101], and one developed in a report by the Johns Hopkins University, Applied Physics Laboratory, [81]. Data from a SDC multi-year study of software development programs, hereafter referred to as SDC Phase I [46], II [132], and III [49] Studies, were used to evaluate the factors affecting cost estimates and to support other relevant analyses. The Phase III data was used extensively because of the large number of observations and variables it contains. The other data bases were used on a selected basis to corroborate results obtained from the SDC data. When quantitative and qualitative data presented in other sources of information (see Bibliography) substantiated the analysis results, these were also identified.

PARMIS Data received from the AF Data Systems Design Center contained baseline estimates of costs and actual expenditures, the type data considered important to the study. However, it included inconsistencies and appeared so unique, it was discarded. Over 85 percent of the programs reported, all of which were developed at the installation, did not exceed budget estimates. This situation made the original estimates suspect. In addition, it could not be determined from the data received whether or not programs had been completed (in cases when the actual costs were less than half that anticipated).

### 3.2 The analysis

The criterion used for identifying the factors that affect software costs was that the probability of correlation existing between the cost and factor must be at least 95 percent. For the analyses, man-months of effort in software development was used in lieu of costs. That is, the effects of factors were assessed in terms of their impact on the man-months of effort required. This was translated into effects on productivity in terms of source instructions per man-month or object instructions per man-month, as appropriate. Man-months was preferred because it avoided

conversion into constant year dollars and the associated problems of cost escalation.

As shown in Figures 1 and 2, the data from the SDC Phase III study was highly dispersed. A major segment of the more highly dispersed data was programs in which the delivered code was 40 percent or less of the total code written. Several factors could cause this effect (e.g., changing requirements, inadequate specifications, inadequate communication, etc., alone or in combination). These programs were deleted from the data set and were analyzed separately. By removing this data, the reports of other factors could be assessed more effectively.

Another source of dispersion was that some programs reflected a productivity so great as to be unreasonable. As an example, one data point reflected 6500 new source instructions delivered for one man-month of effort. The source of error in the data was not known; it may have been caused in recording the data or by recoverable code being reported as new code. The data was assembled prior to 1967, therefore, it did not reflect modern programming practices which reportedly can result in unusually high productivity. To decrease the impact of such errors, data points reflecting a productivity of greater than 2000 source lines per man-month were also removed from the data base. The data plots in Figures 1 and 2 indicate the specific data points deleted. Forty of the 169 sets of data were removed. Discrepancies were also noted when comparisons were made with the same data presented in a previous SDC study. Even with corrections and the deletion of the two sets of data causing dispersion, the data was still highly scattered. Inconsistency in interpreting information requested, and errors in recording the information may account for some of the variance. However, the factors identified and analyzed undoubtedly account for most of the dispersion.

The magnitudes of the effects were assessed using multivariate regression techniques. Several runs were made, adding and deleting combinations of variables in order to develop a better estimate of the

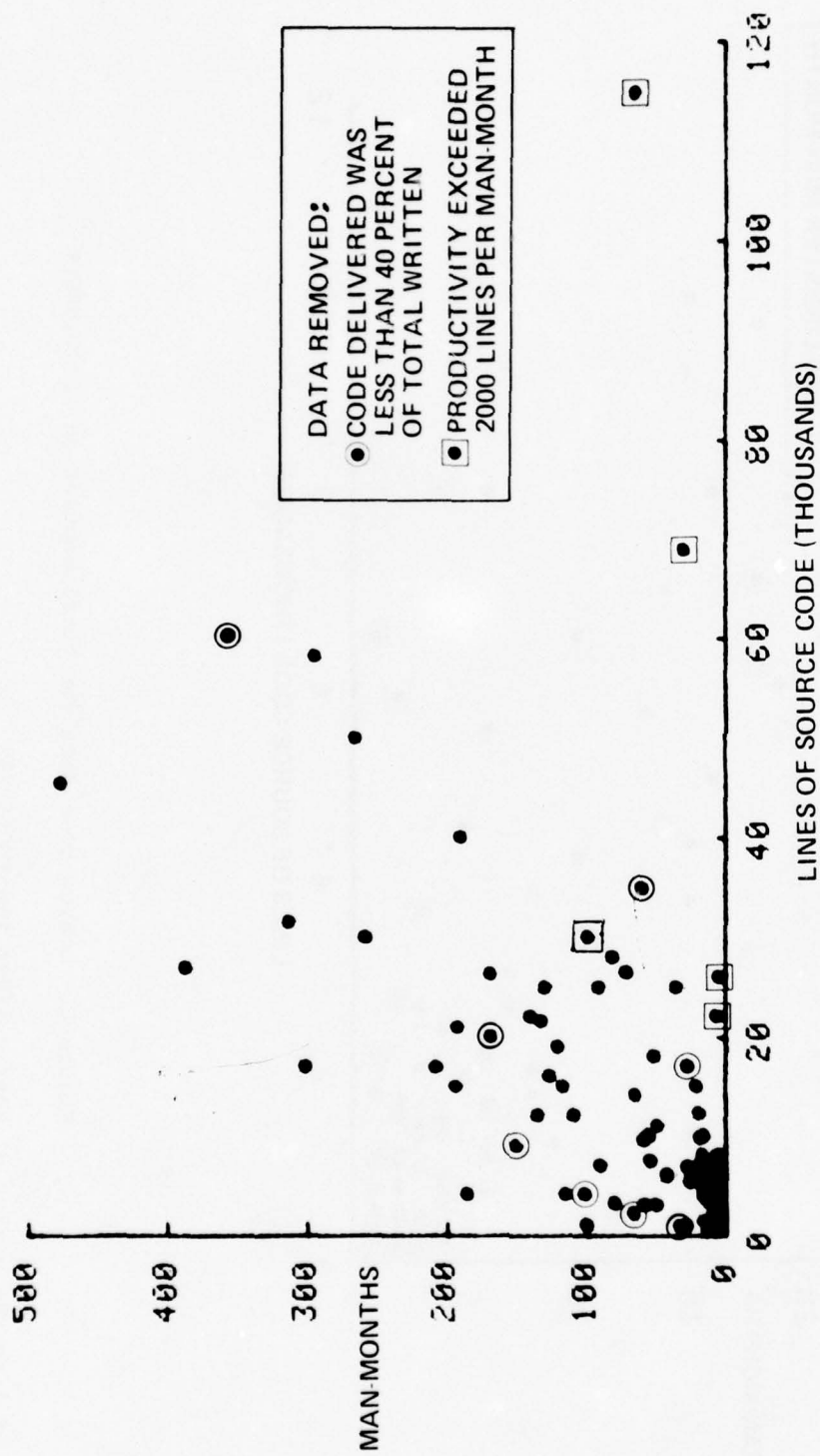


Figure 1. Source code data for total population

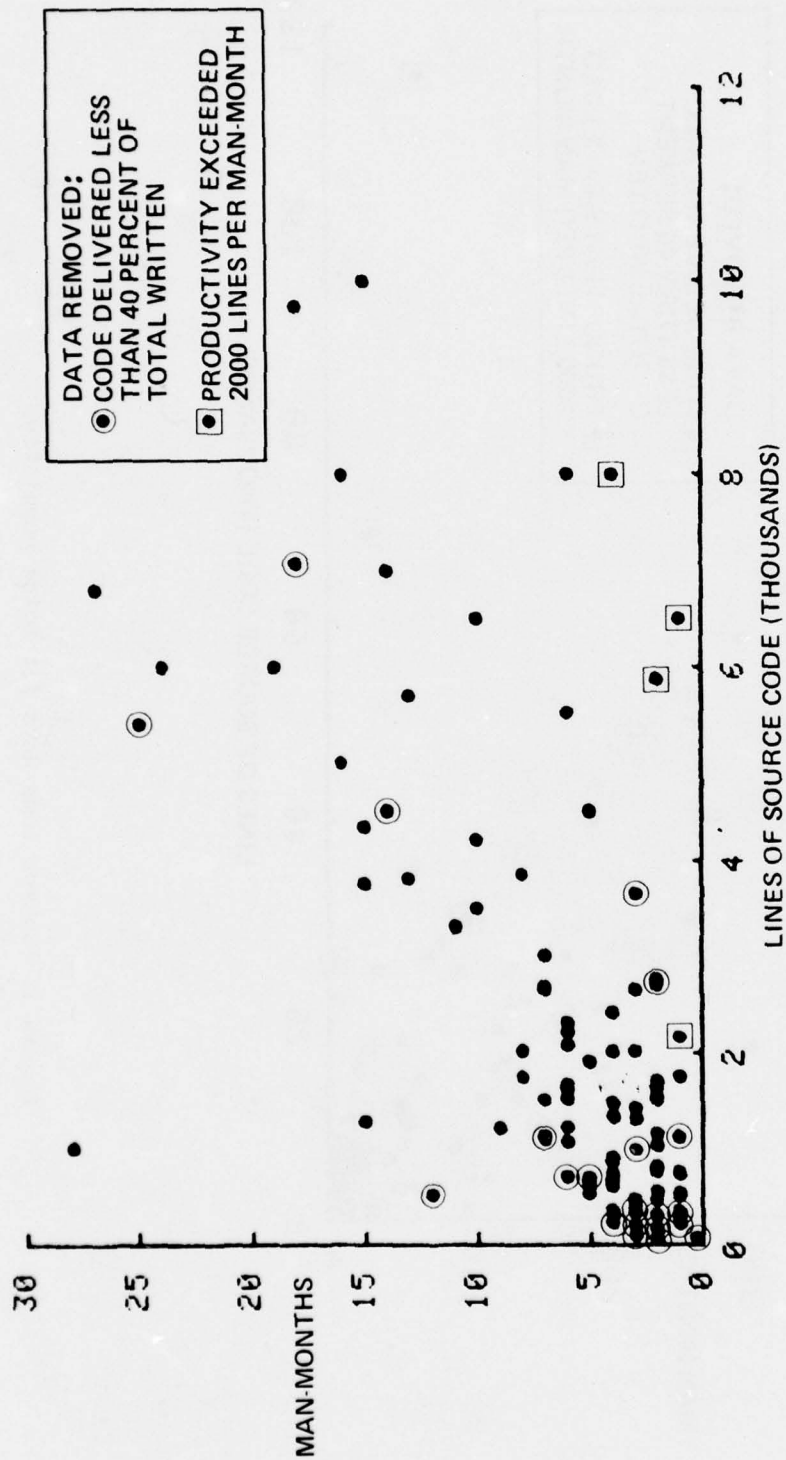


Figure 2. Source code data for total population ( $I < 10,000$ )\*

\* I is the number of source lines instructions.

effect various probable combinations of the factors have on costs. Thus, the values derived for the regression coefficients represent best estimates of the impacts which would be expected to occur. Quantitative information could not be derived on all factors; however, where possible, qualitative information was presented. The high dispersion of the data made it difficult to isolate several likely cause and effect relationships reported in the literature.

This analysis was also performed for subsets of the total population: command and control, utility, scientific and business programs. In the analysis of these subsets, it was found that some variables are not relevant to the entire population. For example, business type programs were the only ones using time share processing in software development.

### 3.3 The results

The study identified over 40 factors which are considered to have a significant impact on software development manpower requirements, productivity, and cost, as appropriate. Those having the greatest effect encompass hardware constraints to the development, environmental parameters detrimental to productivity, software design characteristics, especially when they involve innovations or unique requirements, and the status of the performance specifications. The effects of individual factors are discussed in subsequent paragraphs.

3.3.1 Purchaser domain factors. Table 2 identifies the purchaser domain factors analyzed, and summarizes the results of the analysis. The effects of factors were evaluated in terms of their impact on manpower (man-months), productivity (delivered source instructions per man-month), cost, and other factors.

3.3.1.1 Performance specifications. Factors analyzed as dimensions of software performance specifications include the input and output requirements, data base requirements, operational requirements, response

TABLE 2. PURCHASER DOMAIN

Area	Factor Investigated	Type of data analyzed		Impact	Dependent variable affected	Effect	Comments
		Qual.	Quant.				
Performance Specifications	A. Input Requirements						
	1. No. of Input Message Types	Yes	Yes	Yes	Man-Months	Insignificant	Parameters for this factor are far more important in business data processing applications than C/C or scientific applications. Mainly sizing parameters.
	2. No. of Input Message Items	Yes	Yes	Yes	Man-Months		
	B. Output Requirements						
	1. No. of Output Message Types	Yes	Yes	Yes	Man-Months	Insignificant	Parameters for this factor are far more important in commercial data processing applications than C/C or scientific applications. Mainly sizing parameters.
	2. No. of Output Message Items	Yes	Yes	Yes	Man-Months		
	3. Special Displays	No	Yes	Yes	Productivity	C/C: 10% dec. SC: 10% dec. UT: No effect BS: 30% dec. All: 10% dec.	
	C. Data Base Requirements						More impact on commercial than scientific. Should have impact on command and control, but quantitative data does not support.
	1. No. of Classes of Items in Data Base	Yes	Yes	Yes	Man-Months	Minor	
	2. No. of Words in Data Base	Yes	Yes	Yes	Man-Months	Minor	
	3. Data Management	Yes	No	Yes	Cost	Variable	DBMS has life-cycle cost benefits of design flexibility and reduced maintenance. However, increased development costs are incurred and it makes inefficient use of CPU resources.
	D. Operational Requirements						
	1. Definition	Yes	Yes	Yes	Productivity	C/C: 35% dec. SC: 50% dec. UT: No effect BS: No effect All: 10% dec.	Productivity decreases as definition goes from detail to vague.
	2. Changes	Yes	Yes	Yes	Productivity	Ave: 5% dec. Max: 95% dec.	Magnitude highly dependent on individual case.

Applications:

C/C - Command and Control    BS - Business  
SC - Scientific                    All - All the above  
UT - Utility

TABLE 2. PURCHASER DOMAIN (Continued)

Area	Factor Investigated	Type of Data analyzed Qual. Quant.	Impact	Dependent variable affected	Effect	Comments
Performance Specifications (Continued)	3. Interface to Design	Yes No	Yes	Productivity	Decrease	Adverse impact for requirements analysis which does not provide clear inputs to design.
	E. Response Time Requirements	No	Yes	Productivity (Impact of real-time operation)	C/C: 25% dec. SC: 40% dec. UT: 70% dec. BS: No effect All: 25% dec.	Decrease as requirement goes from once per day to real-time
	F. Hardware Constraints					
	1. Time	No	Yes	Productivity	C/C: 34% dec. SC: 40% dec. UT: 57% dec. BS: No effect All: 34% dec.	Highly correlated with response time. When the constraint is present it generally means real-time
	2. Memory	No	Yes	Productivity	C/C: 20% dec. SC: 20% dec. UT: 15% dec. BS: Undet. All: 30% dec.	Expected impacts for using 65 to 80 percent of available memory. Impact much greater if 80 percent of memory used.
	3. Time and Memory	Yes	Yes	Productivity	58% dec.	
	4. First Time on CPU	Yes	Yes	Productivity	48% dec.	
	5. Concurrent Development of Hardware	No	Yes	Productivity	C/C: 40% dec. SC: 55% dec. UT: 20% dec. BS: 25% dec. All: 45% dec.	
	6. Time at which CPU is Specified in the Schedule	Yes	No	Cost	Variable	The later in the schedule the CPU is specified, the higher the software cost. The major impact, however, is an indirect effect on total system cost.
	G. Documentation Requirements					
	1. Number of External Pages	No	Yes	Man-Months	Unknown	Highly correlated with resource expenditures. Can be used as a confirming predictor
	2. Number of Internal Pages	No	No	Man-Months	Unknown	
	H. Requirement for Innovation	No	Yes	Productivity	Decrease	
Performance Specifications (Continued)	I. Language Requirements	Yes No	Yes	Cost	5 to 1 over spectrum	Maximum possible decrease in cost from pure MOL to fully standardized widely used HOL.
	J. Quality Requirements	Yes	No	Cost	Increase	The higher the quality requirements, the higher the development cost. However, anticipate saving over the life-cycle.

Applications:

C/C - Command and Control    BS - Business  
SC - Scientific                    All - All the above  
UT - Utility

TABLE 2. PURCHASER DOMAIN (Continued)

Area	Factor investigated	Type of data analyzed		Impact	Independent variable affected	Effect	Comments
		Qual.	Quant.				
Performance Specifications (Continued)	K. Reliability Requirements	Yes	No	Yes	Cost	Increase	The more stringent the reliability requirement, the higher the development cost. However, anticipate saving over life-cycle.
	L. Testing Requirements	Yes	No	Yes	Cost	Increase	More testing, the higher the development cost. However, anticipate savings over life-cycle.
	M. Verification and Validation	Yes	No	Yes	Cost	Increase of 20% for independent V&V testing procedures.	ASD has utilized a factor of 20% increase in total software development cost if independent V&V is included in the testing procedures.
	N. Transportability	Yes	No	Yes	Cost	Increase	Mainly a function of language requirements. Excessive transportability requirements with regard to standardization of HOL can reduce flexibility, and thus increase development costs.
	O. Maintenance Requirements	Yes	No	Yes	Cost	Increase	Increase development costs, however anticipate reduced life-cycle costs.
Task Identification Proposal Preparation Program Management	---	No	No	Unknown		Unknown	
	---	No	No	Unknown		Unknown	
A. Development Schedule		Yes	Yes	Yes	Schedule, Productivity, and Time Distribution of Resources	See Curves	Size is highly correlated with schedule length. Men and months are not freely exchangeable, thus productivity is affected. The exact analytical form of the exchange ratio is not known. Historical data is available on time distribution of resources, but shape of distribution is not consistent.
	B. Purchaser Experience	No	Yes	Yes	Productivity	Limited Experience 40% dec. Extensive Experience 27% dec. No Experience No effect	Purchaser experience seems to limit developer productivity. However, the product received may have higher quality.
	C. Number of Management Controls	No	Yes	No	Productivity	No effect	Management controls appear to have no effect on productivity, but may affect quality of received product.

Applications:

C/C - Command and Control      ES - Business  
SC - Scientific                      All - All the above  
UT - Utility

TABLE 2. PURCHASER DOMAIN (Continued)

Area	Factor Investigated	Type of data analyzed		Impact	Dependent variable affected	Effect	Comments
		Qual.	Quant.				
Program Management (Continued)	D. Communications	Yes	No	Yes	Cost	Increase	Ineffective communication should increase development cost.
	E. Planning for System Growth	Yes	No	Yes	Cost	Increase	Impact can be large if entire development is in assembler, and growth outstrips initially specified CPU configured to its maximum capacity.
Development Environment	A. Average Turnaround Time	No	Yes	Yes	Productivity	45% dec. over range	Optimum appears to be 2 computer turns per day. Less than this or greater than this affects productivity in an adverse manner.
	B. Time Sharing vs. Batch	Yes	Yes	Yes	Productivity	Time Share 20% more productive	Time Sharing appears to be about 20% more productive than Batch.
	C. Developer Using Another Activity's Computer	No	Yes	Yes	Productivity	30% dec. in productivity	If the developer is forced to do his development on a computer run by another organization, it appears to reduce his productivity by about 30%.
	D. Programmer access to computer	No	Yes	Yes	Productivity	Limited Computer Access increases productivity 50%	Programmers should not have hands on machine availability in a batch environment. A separate operations staff appears to be more productive.
	E. Development at Operational Site or Not	No	Yes	Yes	Productivity	28% dec. if at Operational Site	If the developer is required to do his development at the operational site, his productivity may decrease as much as 28%.
	F. Development and Target Computer Different	No	Yes	Yes	Productivity	C/C: 55% dec. S/C: 10% dec. U/T: 30% dec. B/S: No effect ALL: 20% dec.	If it is possible to do the development on the same computer that will be used in the operational environment, it should increase productivity significantly.
	G. Number of Development Locations	No	Yes	Yes	Productivity	20% decrease if more than one Development Location.	Advantage to having development at one site, if possible
	H. Development at a Military Location	No	Yes	Yes	Productivity	No Effect	

## Applications:

C/C - Command and Control      BS - Business  
 SC - Scientific                      All - All the above  
 UT - Utility

TABLE 2. PURCHASER DOMAIN (Continued)

Area	Factor Investigated	Type of data analyzed Qual. Quant.	Impact	Independent variable affected	Effect	Comments
Development Environment (Continued)	I. Independent Program	Yes	No	Productivity	No Effect	The quantitative results appear to contradict intuition. It would seem that a software development that does not have to interface with previously developed software would be more productive.
	J. Programming Environment	Yes	Yes	Productivity	Variable	The Programming Environment created by such disciplines as Modern Programming would appear to have a beneficial cost effect.
	K. Support Software	Yes	Yes	Cost	Variable	The availability and quality of support software should have a large impact on development costs.
	L. Programming Facilities	Yes	Yes	Productivity	Variable	The quality and availability of programming facilities such as computer facilities and personnel to operate them, can have a large impact on development cost.
Production Environment	M. Multiple Software Utilization Sites	No	Yes	Productivity	28% decrease for multi-site operation	If the delivered software has to operate at more than one site, then development productivity is reduced
	N. Average Operate Time	No	Yes	Productivity	30% decrease over spectrum	Decreases as one moves toward real-time. Highly correlated with response time.
	O. Average Frequency of Operation	No	Yes	Productivity	30% decrease over Spectrum	Decreases as one moves from once per month to real-time. Highly inter-correlated with above factor and response time.
User Interfaces and Participation	P. Number of User Organizations with which Program must Interface	No	Yes	Productivity	No Effect	Virtually no correlation between number of user organizations and productivity.
	Q. Number of Agencies Whose Concurrence is required on Operational Design Specifications	No	Yes	Productivity	No Effect	Virtually no correlation between number of design concurrence agencies and productivity.
	R. End User Requirements	Yes	Yes	Cost	Variable	The worst case could be a 100% redo of the software if the delivered product completely fails to meet the user's needs.

Applications:

C/C - Command and Control

RS - Business

SC - Scientific

All - All the above

UT - Utility

time, hardware constraints, documentation requirements, quality, language, reliability, testing, verification and validation, maintenance, and transportability.

3.3.1.1.1 Input requirements. The input requirements analyzed (number of input message types and number of input message items) appeared to have little impact on the software costs when analyzed in the presence of other variables (e.g., number of source or object instructions). In commercial or business type data processing, where I/O is generally large relative to computation, these types of factors have been claimed to be determinants of software development cost, [104]; this is especially true for transaction type processing. For scientific processing or command and control applications, these parameters do not appear to be critical.

3.3.1.1.2 Output requirements. Output parameters (number of output message types and number of output message items) were examined with the same results as for the input parameters.

3.3.1.1.2.1 Special displays. These can be considered part of the category of factors which are associated with anything new, unique or innovative in a system. The impact of programming for special display equipment such as plotters or peculiar CRT requirements was analyzed. This factor has the following adverse impacts on development productivity:

- Command and control programs, 10 percent decrease
- Scientific programs, 10 percent decrease
- Utility programs, no appreciable effect
- Business programs, 30 percent decrease
- All programs, 10 percent decrease

3.3.1.1.3 Data base requirements. Two size parameters were examined quantitatively: the number of classes of items and the number of words

in the data base. These parameters have been claimed to be determinants of software development costs for business applications. The analysis, confirming the results of PRC [3], revealed minor impact in commercial applications, and even less significant impact in scientific applications. No quantitative impact was indicated on command and control programs. However, with information storage and retrieval being a major element of command and control applications, this factor would be expected to have a large impact.

The type of data management utilized for permanent files was also evaluated. The basic choices considered an ordinary file management system versus a data base management system (DBMS), of which the latter offers data independence from file organization. A DBMS allows one to alter the format of records within the data base without modifying every program that accesses the data base, since the access programs are independent of the physical organization on the file. The only program that would have to be changed in this case is the DBMS program. A DBMS has two beneficial and two adverse effects relative to a file management system and its effect on software cost. The beneficial effects are that (1) it enhances design flexibility, and (2) it should reduce maintenance costs. The adverse effects are that (1) it increases development costs since the DBMS program has to be written, and (2) a penalty is paid in CPU memory and time usage. For many systems the penalty in CPU efficiency cannot be tolerated. Also, in a number of applications the tradeoff between the life-cycle cost benefit expected using a DBMS, and the associated higher development costs, is complex and requires careful analysis. Quantifiable data to examine these tradeoffs were not available.

3.3.1.1.4 Operational requirements. The impact of requirements analysis on the cost of a software project is extremely critical. Unless the operational requirements are definitized to the maximum extent possible during the conceptual and design phases, continual efforts to

"freeze" the requirements will result in increased costs due to redesign, coding, testing, and schedule slippage. These costs could become significant, particularly in large programs that have interacting modules. Unfortunately, there is very little quantitative data with which to analyze the impact of this factor.

3.3.1.1.4.1 Definition. The sparse quantitative data used to assess the impact of this element was taken from the SDC Phase III study [49]. The definition of operational requirements given to the developer was divided by SDC into three categories: in detail, in outline, or vaguely. Analysis indicated that productivity (source instructions per man-month) declined as the definition of operational requirements became more vague. The following effects were noted in going from detailed to vague requirements:

- Command and control programs, 35 percent decrease
- Scientific programs, 50 percent decrease
- Utility programs, no appreciable effect
- Business programs, no appreciable effect
- All programs, 10 percent decrease

3.3.1.1.4.2 Changes. Changes in the Operational Requirements are perhaps the most important element affecting software development costs. Very few, if any, projects go through the entire development phase without requirements changing. Often, large amounts of software in various stages of development are discarded because of continually changing requirements. In projects where significant changes in operational requirements have occurred, software productivity measured in terms of the delivered product will be very low. An example of this is the Army Patriot (formerly SAM-D) missile. Over nine years in development, about \$90 million had been spent through 1975 in developing a software package that only contained 250,000 object words, [9]. This yields an object word productivity of 12 instructions per man-month, a

factor of 22 below that normally expected for this size program. Analysis of the data available for this study indicated that on the average changing requirements decrease productivity 5 percent. However, in extreme cases, the effect can result in very low productivity.

3.3.1.1.4.3 Interface to design. Another element that clearly has an impact is the interface between requirements analysis and design. If the requirements analysis does not provide definitive and easily understandable inputs to design, an adverse impact on development cost can be expected. Quantitative data was not available to measure the impact of this element.

3.3.1.1.5 Response time requirements. Although a subset of operational requirements, the response time requirements have such a clearly defined impact that it was treated separately. As requirements change from developing a software module for management reporting that may have to respond perhaps once per day or once per month to developing a module to analyze sensor data in real time, software productivity decreases. The results of the study indicated that real-time operation has the following effects on productivity:

- Command and control programs, 25 percent decrease
- Scientific programs, 40 percent decrease
- Utility programs, 70 percent decrease
- Business programs, no effect, not usually operated in real time
- All programs, 25 percent decrease

3.3.1.1.6 Hardware constraints. In certain applications, hardware constraints have a large impact on software costs. As an example, because of weight and volume constraints of on-board avionics, the amount of memory in the CPU is generally limited. This usually requires that the software be written with extremely efficient memory usage, which lowers software development productivity. There are a number of elements

which have been examined for this factor-CPU constraint in the time domain, CPU constraint in the memory, the combined time and memory CPU constraints, the constraint of first development on the CPU, concurrent development of other components which interface with the CPU, and the time in the development when the CPU is specified.

3.3.1.1.6.1 Time. This element refers to the availability of CPU time to perform any particular function, which becomes a very critical element in many real-time or on-line applications. The data defined time as an element in the software development which was either stringent or readily available. The answers were coded yes or no. For developments where the constraint was present, productivity was reduced by the following:

- Command and control programs, 34 percent decrease
- Scientific programs, 40 percent decrease
- Utility programs, 57 percent decrease
- Business programs, No effect
- All programs, 25 percent decrease

3.3.1.1.6.2 Memory. This element refers to the constraint to software development imposed by the size of the memory of the processor. The analysis revealed the average impacts on productivity as follows:

- Command and control programs, 20 percent decrease
- Scientific programs, 20 percent decrease
- Utility programs, 15 percent decrease
- Business programs, not able to assess
- All programs, 30 percent decrease

There is evidence that the impact on productivity could be several times that indicated [18] as the memory used exceeds 80 percent of that available.

3.3.1.1.6.3 Time and memory. The combined effects of both time and memory constraints were analyzed. The impact in the presence of both constraints can reduce software productivity by the amounts reflected above. Other investigations of the impact of this element have been carried out by Barry Boehm, [18], and E.N. Dodson, [43]. Boehm's results, presented in Figure 3, reflect a drop of 3 to 1 in productivity for worst case conditions. The Dodson results measured the decrease on productivity as 5.2 to 1.

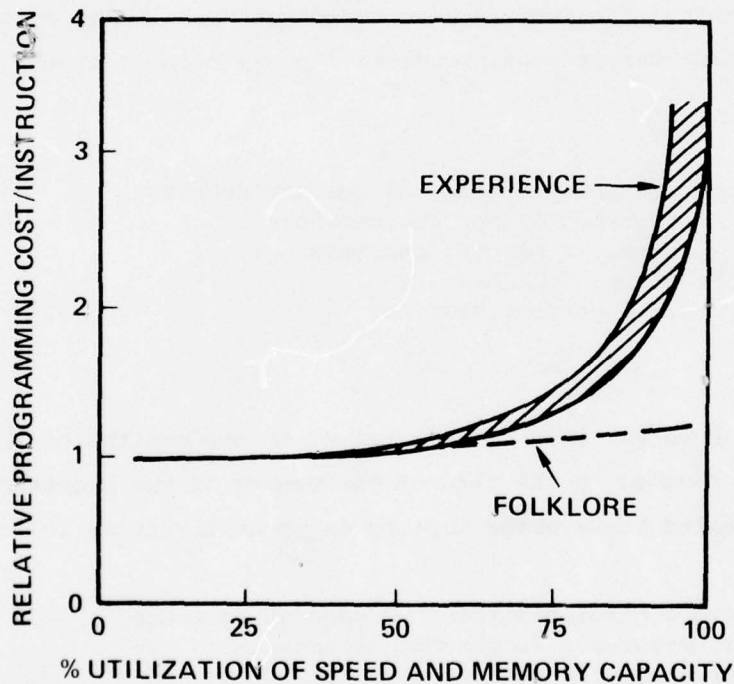


Figure 3. Effect of time and memory constraints on software productivity (developed by Barry Boehm [18])

3.3.1.1.6.4 First development. This element reflects a new CPU for developing the code or for which the code is targeted, or the first time the developer has worked with either the development CPU or the target CPU. If either the development or target CPU is new, then additional support software will generally have to be developed along with the operational software. This is especially true if the development CPU is new. If the developer is new to an existing CPU, either development or target, then one expects his normal productivity to be affected in an adverse manner. From information presented by Halstead, [61], one could infer that developer personnel would require six months to become fully productive on a CPU unfamiliar to them. The analysis provided a measure of the impact on productivity of a CPU which is unfamiliar to the software developer. When the constraint was present, productivity dropped by 48 percent. This element falls into that category of adverse effects which are experienced when anything new or innovative is involved in the development.

3.3.1.1.6.5 Concurrent development of hardware. Another effect reflecting the development of anything new or innovative, this element includes the concurrent development of any devices, such as receivers, sensors, or ADP peripherals, which must interface with the CPU in the operational environment. Analysis of the data relative to the impact of concurrent development indicated that productivity was reduced as follows over those cases where it was not present:

- Command and control programs, 40 percent decrease
- Scientific programs, 55 percent decrease
- Utility programs, 20 percent decrease
- Business programs, 25 percent decrease
- All programs, 45 percent decrease

3.3.1.1.6.6 Time at which CPU specified in the schedule. In many large weapon systems developments, software development is a critical

path event, since major efforts on the software cannot commence until the source selection for hardware has been completed, [18]. Removing software from the critical path by specifying hardware early in the overall system development, or making software development hardware independent, probably only has a minor impact on software development costs. However, the indirect effect of having software on the critical path could have a large impact on total weapon systems development cost. There was no quantitative data available to measure the magnitude of this effect.

3.3.1.1.7 Documentation requirements. Even though the literature purports that the amount and kinds of documentation vary depending on the type of software [68], and thus should be a factor considered in developing a reliable cost estimate, the impact of these requirements could not be determined from the available quantitative data. Interestingly, the data demonstrated a consistency in the amount of documentation produced as a function of the size of the development effort.

3.3.1.1.7.1 Number of external pages. Analysis of the data indicated that the number of external pages of documentation produced and delivered with the product is practically as good an estimator for resource expenditures as the number of source statements delivered. They are both size estimators, but pages of documentation would not be used as a predictor variable. Pages of documentation correlate highly with man-months on a geometric scale, starting out at six pages per man-month for small projects and decreasing to three pages per man-month for large projects. Thus, it is not surprising that pages of documentation is also highly correlated with source statements. General Research Corporation (GRC) in their study of "Life-Cycle Costing of Major Defense System Software and Computer Resources" for ESD, Hanscom AFB [59], arrived at similar results.

There is also a mildly strong correlation between the amount of documentation produced and software productivity, where the amount of documentation is defined as the number of pages per 1000 source instructions delivered. As the amount of documentation increases by a factor of 10, productivity decreases by 63 percent. For example, if productivity is 320 instructions per man-month with 10 pages per 1000 lines, then productivity will be 120 instructions per man-month with 100 pages per 1000 lines.

Accordingly, estimators developed using these relationships could be utilized to determine the amount of documentation expected for a software development, but the relationships should not be used as an estimator of man-months.

3.3.1.1.7.2 Number of internal pages. The number of pages of documentation produced by the developer for internal use during development has virtually no correlation with resource expenditures. This type of documentation obviously varies greatly among developers and should not be used as an indicator of resource expenditures.

3.3.1.1.8 Requirement for innovation. Any requirement that involves innovation or the development of unique software, such as new approaches to tactical problems, generally has an adverse impact on cost and a severe decrease on productivity. The magnitude of the effect could not be assessed from available quantitative data.

3.3.1.1.9 Language requirements. Most of the impacts for this factor have to be treated in a qualitative manner. It is generally accepted that this factor has an immense impact, not only on development cost, but perhaps more importantly, on life-cycle cost. The major choice is between a High Order Language (HOL), such as FORTRAN and JOVIAL, which are essentially machine independent, and a Machine Oriented Language (MOL), which is machine dependent. The major advantages of an HOL are:

- easy transportability of code from one CPU to another, and
- the expansion ratio.

The expansion ratio is a measure of the amount of machine code generated per source statement. For a large main frame, with an optimized compiler, it is about four machine words per source statement for most languages for a typical program. For most assemblers, the ratio is relatively close to (1 to 1). Since programmers work at about the same rate in source statements, independent of source language, the cost impact of HOL versus MOL is 4 to 1. For minicomputers, with their rather primitive machine language instruction sets, the impact is even greater, since the expansion ratio is about 20 to 1. As maintenance rates also tend to be constant relative to source code, independent of language, there is a major life-cycle benefit to HOL versus MOL. The huge cost benefits due to the expansion ratio of an HOL cannot always be realized.

The major disadvantages of an HOL are:

- the compiler usually generates inefficient code in the time and memory domains relative to the average assembler programmer; and
- certain types of operations, such as bit manipulations, are sometimes hard to implement using HOL.

The disadvantage which has the most cost impact is inefficiency of the generated machine code. The inefficiencies of generated code range from 1.4 to 1 (40 percent more machine code and taking 40 percent more time to execute for the HOL) for the large sophisticated main frame with a highly optimized compiler, to 4 to 1 for the most primitive of minicomputers with a highly unoptimized compiler. This gives effective expansion ratios of 2.85 to 1 for the worst case, to 5 to 1 for the best case, of HOL versus MOL. In many cases, such as for real-time

applications or for on-board avionics with severe memory constraints, these inefficiencies cannot be tolerated, leaving HOL as a non-viable alternative.

Therefore, many variables must be considered when assessing the cost impact of language requirements established by the purchaser for adherence by the developer. Among them are:

- the language chosen;
- the target machine for the compiler output;
- the optimization of the compiler chosen for the language selected;
- the degree of standardization in the compiler chosen, which affects transportability of source code;
- the validation of the compiler chosen; and
- the degree to which functions, that are to be implemented with HOL, can meet the efficiency requirements dictated by the application.

Thus, it is not simply a matter of dictating HOL for all functions to be performed by software in a given application, and reaping the expansion ratio and transportability benefits. For some functions in some applications an HOL implementation will fail, because of the inefficiency of the generated code. For the best case, in which it is specified that all source code is to be generated with an HOL, the maximum cost benefit that can be gained over an entire MOL implementation is 5 to 1 over the life cycle of the system. One could possibly do better than this if there is a CPU change over the life cycle, and code has to be transported from one CPU to the other.

3.3.1.1.10 Quality requirements. The elements comprising quality of software has been a subject for discussion since the advent of computers. Literature contains various definitions of quality and identifies attributes of software that should assure high quality products.

Finally, under an RADC contract, the General Electric Company, [107], has defined software quality as consisting of 11 characteristics:

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability
- Maintainability
- Testability
- Flexibility
- Portability
- Reusability
- Interoperability

Some of these attributes are discussed in succeeding paragraphs. Including criteria for each of these characteristics into a performance specification will undoubtedly decrease development productivity. However, before the criteria are specified, tradeoffs should be made to assess the impact on the software life-cycle costs. For this study, there was no quantitative data available with which to assess the impact of quality requirements.

3.3.1.1.11 Reliability requirements. Reliability is one of the attributes of software quality but it is such an important attribute that it was evaluated as a separate factor. The more reliability that is designed into software, the lower the development productivity since more testing will be required to assure that the reliability has been attained. However, over the life cycle of the software, reduced maintenance could result in lower costs. There was no quantitative data available to measure these impacts.

One of the major problems in attempting to measure the effect of reliability is that there is no agreed upon definition of software reliability. Richards, [107], suggests that the definition describe software reliability as the "extent to which a program can be expected to perform its intended functions with required precision." The literature search did not reveal any adequate effort to measure the relationship between reliability and cost.

3.3.1.1.12 Testing requirements. Software development costs are related directly to the amount of testing accomplished during development. The testing requirements also impact software life-cycle costs. Therefore, tradeoffs should be considered to assess the effect of the testing requirements on the life-cycle costs of software. On the average, testing accounts for about 40 percent of development cost. Rates as low as 30 percent and as high as 60 percent, however, are not unusual. One of the major determinants of testing cost is the test plan imposed on the development by the purchaser. Manley, in an article, "Embedded Computer System Software Reliability", in the Defense Management Journal, October 1975, [32], has postulated a cost model for the entire testing process. The genesis of the model is illustrated in Figure 4. There was, however, no quantitative data made available with which to evaluate the parameters of Manley's or any other suggested models of the testing process.

3.3.1.1.13 Verification and validation. One of the major contributors to testing costs in software development is verification and validation (V&V). A major cost consideration in verification and validation is whether or not it is performed by an independent organization. More and more large software development projects are utilizing independent V&V; one of the reasons for this is to improve the quality of the product. Independent V&V will increase development costs, but cost benefits are likely to be obtained over the life cycle. There is no available quantitative data on how much software cost is increased with independent V&V, but a factor in some use at Aeronautical Systems Division (ASD) is that it increases total software development costs by 20 percent.

3.3.1.1.14 Transportability requirements. These requirements are not normally specified in the performance specifications; however, in the event a purchaser subsequently desires to have the flexibility of moving code due to an anticipated change of CPUs over the life cycle, or the possible transfer of code to other software programs, transportability

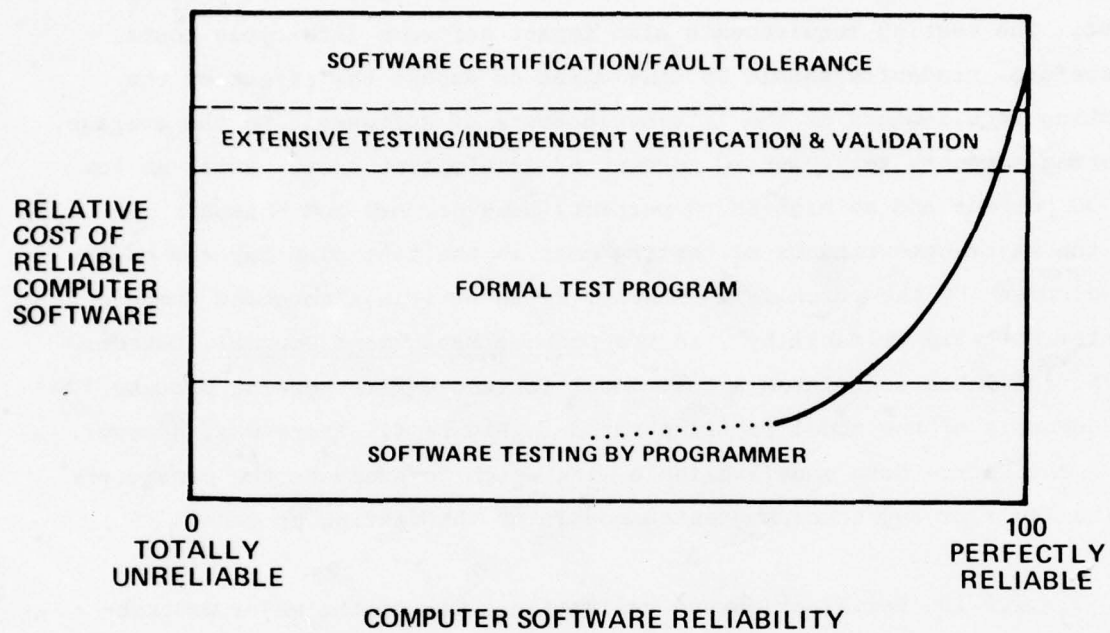


Figure 4. Software reliability economic considerations  
(as proposed by Manley [32])

requirements should be delineated to the maximum extent possible. The degree to which the purchaser will be able to control transportability is mainly a function of the language requirements established. Most of the cost impacts of language requirements were discussed previously; however, they are discussed here in terms of their effect on transportability. The major elements affecting transportability are:

- (1) the type of language chosen, HOL or MOL;
- (2) the number of target machines for which a specific language compiler is available;
- (3) the number of people trained in the language, which is partially related to the above element;
- (4) the degree of standardization in the compiler chosen, for the language selected; and
- (5) whether the compiler chosen allows direct assembler language coding for the target machine of the compiler.

The element having the major impact is Item (1). Items (2) and (3) depend on individual cases, such as the availability of trained personnel and a compiler for the machine into which the code is to be transported. Thus, in a general sense, FORTRAN IV is certainly much more transportable than ALGOL 60. With regard to Item (4), computer vendors have a peculiar penchant for adding non-standard features to standard languages to make their version of the language more flexible. As the number of non-standard features increase, the less transportable the source code becomes. Relative to Item (5), any compiler that allows the programmer to conveniently enter assembler code is inherently less transportable.

Transportability is a cost that should be considered if there is a possibility of changing CPUs over the life cycle or if transfers of code to other software programs are anticipated. With a weapons system life cycle of about 10 years and an average CPU life of about 6 years, this could be a serious consideration. Another facet of transportability is the development cost tradeoff between standardization and flexibility.

If one is required to utilize standard versions of languages, this would probably result in the writing of more code than if one could employ non-standard features. Quantitative data to measure the magnitude of this impact was not available for this study.

3.3.1.1.15 Maintenance requirements. Several factors discussed previously have an important impact on maintenance costs. Most of these factors, which are tacit impositions of maintenance requirements, have an adverse impact on development costs. The exception is imposing the use of HOL languages. Other factors not discussed previously, which are maintenance requirements, have an impact on development cost. Most of these are in the area of documentation, such as:

- requirements for developing run sheets for the programs,
- requirements for developing recovery procedures in the event of software or hardware failure, and
- development of an Integrated Logistics Support (ILS) plan for maintaining the software.

The degree to which items of this nature are implemented would have an adverse impact on development cost, but a beneficial effect on life-cycle costs. No quantitative data to measure the magnitude of this impact was available for this study.

3.3.1.2 Task identification. The adequacy of the definitions for the tasks to be performed during software development, such as the creation of algorithms or definition of data base elements, could have an impact on costs. However, neither qualitative or quantitative information was available to investigate the impact of factors from this area.

3.3.1.3 Proposal preparation time. Neither qualitative nor quantitative information was available to investigate the impact of factors from this area.

3.3.1.4 Program management. Since software development is mostly a labor intensive effort utilizing to a large degree the creative talents

of personnel, program management by the purchaser has to have a significant impact on software cost and cost estimates. It is difficult to delineate what management factors affect these costs and cost estimates, and even more difficult to estimate the quantitative impact of the factors. Some factors have been investigated. The selection of each factor was primarily based on the amount of qualitative and quantitative information that was available. By far the most important, relative to cost impact, is the time period allocated to the software development.

3.3.1.4.1 Development schedule. An adequate amount of quantitative data was not available to analyze the cost impact of the schedule imposed on the developer by the purchaser. Both Brooks, [22], in his classical "The Mythical Man-Month" article, and Aron, [6], have investigated this impact in a pseudo-quantitative manner. The gist of both of these investigations is that men and months cannot be interchanged freely. Both Brooks and Aron imply that schedule is one of the truly important development cost factors. The available quantitative data does, however, enable one to investigate the relationship between size and development time. In addition, there is quantitative data with which to investigate the distribution of resources over the schedule.

3.3.1.4.1.1 Development time. Figure 5 illustrates the relationship between average development time and program size based on historical data in the SDC Phase II study [132]. The curve was developed from the average development times in the historical data; however, the curve is probably indicative of the estimated minimal development time required because it is usually planned that software development be completed within a specific time frame dictated by other considerations. The amount of constraint placed upon the development by specifying the time is not known. It is probable that more time would be required for optimal development, especially for large development programs. Preliminary results of a study by DAI indicate that the average time should be extended for programs with

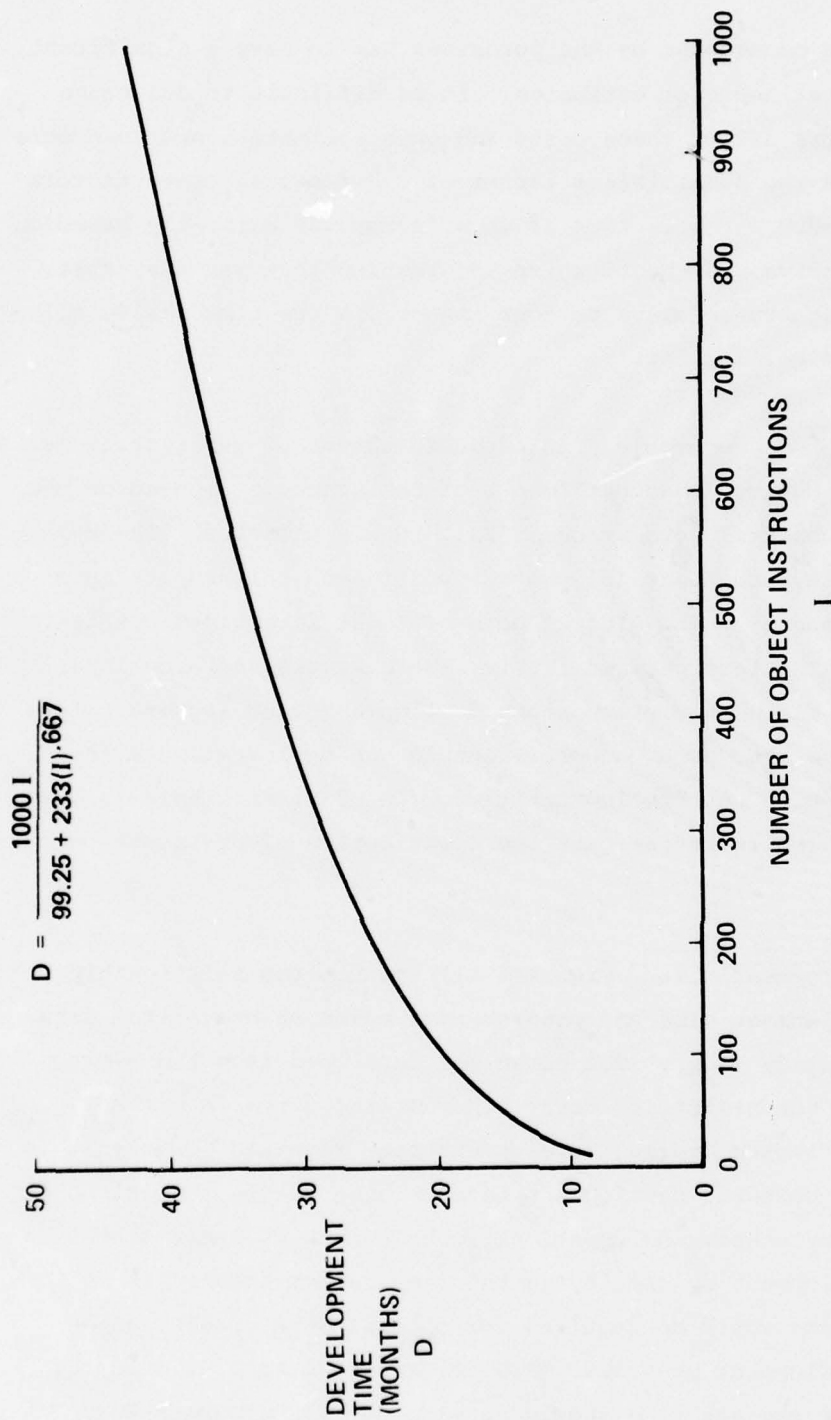


Figure 5. Software development time estimator

greater than 20,000 source instructions, and shortened for programs with less than 20,000 source instructions.

Although both Brooks and Aron expect a significant cost impact for deviation from an optimum schedule, there is no quantitative data with which to measure this impact.

3.3.1.4.1.2 Time-phased distribution of resources over the development time. Aron, [6], and Putnam, in his discussion of the macro-estimating methodology in the ASD Comptroller Automation Study, [1], have postulated the distribution of resources over development to be similar to the curve in Figure 6. Aron's is based on his experience in observing several IBM software developments, and Putnam's is based on analytical fits to a number of Army ADP developments. Devenny's investigation of several ESD software developments, [40], indicates that funding profiles are essentially flat (i.e., level funding), rejecting Aron's and Putnam's hypothesis. Aron, however, indicates that the distribution will flatten out toward level funding with the imposition of Modern Programming.

Putnam has structured his results in the form of an analytical model, which requires the user to indicate the time required to reach peak resource utilization and the total resources expected to be expended in development. The model produces a time-phased distribution of the resource expenditures. There is no indication that this is a recommended utilization of resources, it is just that historically resources were distributed in this manner. However, Putnam does indicate that if the development deviates more than 25 percent from the expected distribution, the development could be in trouble.

An interesting point to Devenny's observations, [40], is that any misestimation of cost is directly reflected in the schedule. For example, if one initially underestimates the cost by a factor of 2, it will take approximately twice as long to complete the development as originally



estimated. Indeed, this seems to be the case with many ESD developments. The cost estimate usually remains frozen, with level expenditure rates, until total expenditures become close to the original estimate, then the estimate is raised and expenditures continue at the same rate until the amended estimate is approached. This scenario is usually repeated until the development is completed.

In summary, there is considerable data available regarding historical relationships of program size to development time, and the distribution of resources over the development time, but there is virtually no data on the cost impact of deviating from these historical norms.

3.3.1.4.2 Purchaser experience. Analysis of data, in which purchaser experience was quantified with a three-state variable: none, limited, and extensive, indicated that limited experience would decrease productivity by 27 percent and extensive experience by 14 percent, and no experience caused no effect. One could infer that if the purchaser has any experience, he tends to exert control over the developer, perhaps limiting productivity. On the other hand, a purchaser with no experience may tend to feel naive about the process, and thus not inhibit the developer. Unfortunately, there was no information available as to the quality of the product associated with purchaser experience.

3.3.1.4.3 Number of management controls. The imposition of management controls should have some impact on the development process, if not in the area of cost, at least in the quality of the product delivered. The former effect was investigated by analysis of data and showed no impact; however there is no quantitative data available relative to the latter. From 11 management controls identified in a questionnaire, responders were asked to indicate the number of controls that were implemented during development. The hypothesis tested was, "The imposition of management controls increases software development productivity." There was virtually no correlation between the number of management controls imposed and productivity.

Nothing, however, could be said about quality of the delivered product. Even if there is no direct development cost benefit to the imposition of management controls there would be a life-cycle benefit if the quality of the received product is enhanced.

3.3.1.4.4 Communications. One would expect the effectiveness with which the purchaser communicates with the developer to be an important cost factor. There was, however, no quantitative data available with which to assess the impact of this factor.

3.3.1.4.5 Planning for system growth. In a number of past developments, software quickly outstripped the capacity of the initially specified hardware. This usually occurred because requirements were increased or changed during development, and because of the tendency to underestimate the size of the required software. The worst impact would be in a situation where the entire development is in assembler, and it was suddenly determined that the target CPU, configured with its maximum possible memory and all available firmware, could not accomplish the stated requirements, thus requiring a change to a CPU with a different architecture. In this case, the entire software program package would have to be redesigned and recoded. At the other extreme, the minimum impact would occur in the case where growth could be accommodated simply by increasing the amount of memory in the CPU originally specified. The ability to accommodate growth is obviously impacted by language requirements and transportability requirements, as discussed earlier. Developing a management plan to accommodate growth involves a comparatively small cost. However, if growth occurs and no plan exists, the impact on software development cost could be appreciable.

3.3.1.5 Development environment. The environment in which the developer is forced to work has a significant impact on software development. Part of this environment is created by the developer, but it is a

responsibility of the purchaser to ensure that a productive environment is created and maintained for software development. As an example, major problems that could occur with secondary resources might be the unavailability of machine time or publications support at appropriate times. Another major problem might be the amount and quality of support software available to the developer. Quantitative data was available for a number of factors from this area which permitted measurement of their impact on cost; others had to be treated qualitatively.

3.3.1.5.1 Average turnaround time. This factor relates to the availability of machine resources to the programmer; quantitative information was available for the analysis of its impact. Turnaround time is the amount of time required for a programmer to receive his results once he submits a run to a computer facility. The factor generally only has significance in a batch environment. The turnaround times reported in the data were divided into four categories: less than 2 hours, 2 to 11 hours, 12 to 24 hours, and greater than 24 hours. Analysis produced a range of effects on productivity of 45 percent for the four cases. The least impact occurred with turnaround times of 12 to 24 hours. The greatest impacts occurred with turnaround times of less than 2 hours, followed by turnaround times of greater than 24 hours. Thus, both too little or too much computer availability appear to be counter-productive. Based on this analysis the optimum is to provide programmers with computer services, i.e., turnaround, twice a day.

3.3.1.5.2 Time sharing versus batch. Available quantitative data seems to indicate time sharing is more productive than batch processing by about 20 percent. Independent results by Sackman, as cited by Boehm, [16], produced an almost identical answer.

3.3.1.5.3 Developer using another activity's computer. In many instances, especially in weapons systems development, the government may

supply the computer as well as the operations personnel for the machine on which the software is to be developed. Analysis of quantitative data indicates that this situation reduces software productivity by 30 percent. The ordering in descending effect would be utility, scientific, command and control, and business (almost no effect).

3.3.1.5.4 Programmer access to computer. Analysis of quantitative data indicates that if programmers are required to submit runs to computer operators for processing rather than having the programmers accomplish the computer runs themselves, productivity is improved 50 percent.

3.3.1.5.5 Operational site development. If the developer is required to develop the software at the operational site, he appears to have a distinct disadvantage, since it decreases productivity 28 percent.

3.3.1.5.6 Development and target computer different. The target computer (i.e., the target of the software development) is the computer that will be employed at the operational site. In many software developments, especially weapons systems, the development computer is frequently different from the target computer. This difference can decrease productivity as follows:

- Command and control program, 55 percent decrease
- Scientific program, 10 percent decrease
- Utility program, 30 percent decrease
- Business program, no appreciable effect
- All programs, 20 percent decrease

It is recognized that in many cases it is not feasible to develop the software on the designated target computer. This is especially true for small on-board avionics computers which have little, if any, support software. However, if feasible, it appears prudent to have the development and target computer the same.

3.3.1.5.7 Number of development locations. Analysis of quantitative data regarding the number of locations at which portions of a software package are being developed indicates that productivity decreased by 20 percent if development is accomplished at more than one location.

3.3.1.5.8 Development at a military location. Based on the analysis of available quantitative data, software development at a military location has essentially no effect on productivity.

3.3.1.5.9 Independent program. Independent program means the software is not required to interface with any other software programs. Analysis of quantitative data indicates that this factor has virtually no effect on software cost. One would assume that developing an independent program would be more productive; however, the analysis of the available quantitative data did not support the assumption.

3.3.1.5.10 Programming environment. Certain modern programming techniques which affect programming environment, such as Chief Programmer Teams and Programming Support Libraries, should have a beneficial effect on software development cost. However, there was inadequate quantitative data with which to measure the impact of this effect. Aron, [6], estimates that the imposition of modern programming techniques should decrease cost by 40 percent over the same development using more traditional techniques.

3.3.1.5.11 Support software. The availability, quality, and, if applicable, the concurrent development of support software, such as a new language, can have an enormous impact on software development costs. Quantitative data to measure the impact of this factor, however, was not available.

3.3.1.5.12 Programming facilities. The development computer facilities along with the personnel to operate the installation are an important

development cost factor. The quality of the support software can have an appreciable effect on these costs. Here again, there was no data available to quantify the impact of this factor.

3.3.1.6 Production environment. Certain factors involved in the area of production environment have considerable impact on development productivity.

3.3.1.6.1 Multiple software utilization sites. Analysis of quantitative data indicated that development productivity decreases if there is a requirement for the delivered software to operate at more than one location when it gets into the field. From the analysis, it appears that developing for a multi-site operation reduces productivity by 28 percent.

3.3.1.6.2 Average operate time. Quantitative analysis of this factor indicates a very high correlation with the response time variable in performance specifications. They both address the same characteristic of the system. As average operate time goes from greater than one hour toward real-time, development productivity decreases by a factor of 30 percent.

3.3.1.6.3 Average frequency of operation. Analysis of this factor also indicated a high correlation with the response time variable and with the average operate time. As one moves from software that only has to run once per month to software that has to operate in real-time, software productivity decreases 30 percent.

3.3.1.7 User interfaces and participation. This area is primarily associated with the quality attribute of usability. User requirements should be definitized to the maximum extent possible and incorporated in the conceptual and design phases of software development to readily assure user acceptance. There were two factors for which quantitative information was available in this area, and one other factor that was treated qualitatively.

3.3.1.7.1 Number of user organizations with which program must interface. Analysis of available quantitative data demonstrated that as the number of user organizations with which the developed software must interface increased, there was no effect on software development productivity.

3.3.1.7.2 Number of agencies whose concurrence is required on operational design specifications. Again, the analysis of available quantitative data demonstrated that as the number of agencies that have to approve operational design specifications increased, there was no effect on productivity.

3.3.1.7.3 End user requirements. The extremes regarding this factor illustrate the potential impact on software development. At one extreme there might be the user who is not involved in developing requirements, and yet accepts the delivered product without exception. In this case there is no cost impact. At the other extreme there might be the user who is completely involved in requirements analysis, and yet rejects the delivered product as not totally acceptable to him, requiring a redo. This could happen, for example, if the requirements were only broad statements obtained from the user and not definitized. The result could be a system that is difficult to operate and maintain as a result of inadequate design and development. Accordingly, analysis of the cost impact is very difficult without additional quantitative data. Boehm, [16], cites a couple of examples of 67 percent and 95 percent rewrites of software because user requirements were not defined properly.

3.3.2 Developer domain factors--preparation procedures. Table 3 identifies the factors analyzed, and summarizes the results of the developer domain. As in the purchaser domain, most of the effects have been evaluated in terms of the impact on productivity or cost.

3.3.2.1 Base lines. The configuration documentation describing the hardware and software to be utilized in a specific project could have

TABLE 3. DEVELOPER DOMAIN PREPARATION PROCEDURES

Area	Factor Investigated	Type of data analyzed		Impact	Independent variable affected	Effect	Comments
		Qual.	Quant.				
Base Lines	A. Participation of Programming Organization in Requirements Analysis	No	No	Unknown	Cost	Unknown	Productivity increases with increased participation.
	B. Misunderstanding Requirements	Yes	No	Yes	Cost	Variable	Could require correction of software program ranging from minor change to a complete redo.
	C. Failure to meet Understood Requirements	Yes	No	Yes	Cost	Variable	Impact could vary significantly depending on circumstances.
Identifying Project Tasks	A. Number of Project Tasks	Yes	Yes	Yes	Man-Month	Variable	Mainly a sizing parameter. If task is equated with module, then there is a high correlation with man-months.
	A. Complexity	Yes	Yes	Yes	Productivity	Unknown	Has to have an effect, but no successful rating scales exist.
Design	B. Stability	No	Yes	Yes	Productivity	50% decrease over spectrum of no design changes to complete redesign.	
	C. Modern Programming Techniques	Yes	No	Yes	Productivity	67 percent increase	Estimate by Aron of IBM
Software Sizing and Cost Estimation Techniques	D. Design Techniques	Yes	No	Yes	Productivity	Unknown	Inconclusive findings
	E. Flexibility	Yes	No	Yes	Cost	Variable	Has an adverse impact if requirements remain static, beneficial if requirements change.
	A. Cost per Instruction Method				Variance of Actual to Estimated		These techniques only affect the estimate, not the actual cost.
	1. Sizing Estimate Error	Yes	No	Yes	Cost	See Table 6.	
	2. Definition of Instruction a. Total Object Words	Yes	Yes	Yes	Cost	Std. Error: Upper Band = 182% of estimate Lower Band = 35% of estimate	

Applications:

C/C - Command and Control    BS - Business  
 SC - Scientific                    All - All the above  
 UT - Utility

TABLE 3. DEVELOPER DOMAIN PREPARATION PROCEDURES (Continued)

Area	Factor Investigated	Type of data analyzed		Impact	Dependent variable affected	Effect	Comments
		Qual.	Quant.				
Software Sizing and Cost Estimation Techniques (Continued)	b. Total Object Words Minus Data Areas	Yes	Yes	Yes	Cost	Std. Error: Upper Band = 160% of estimate Lower Band = 38% of estimate	An estimate based on delivered code will be poorer, but it may be the only size measure available.  Support software sizes will reduce the error in the estimate if support software development is in software cost.
	c. New Object Words Minus Data Areas	Yes	Yes	Yes	Cost	Std. Error: Upper Band = 140% of estimate Lower Band = 42% of estimate	
	d. New Source Lines	Yes	Yes	Yes	Cost	Std. Error: Upper Band = 68% of estimate Lower Band = 60% of estimate	
	e. Delivered Code vs. Total Code	Yes	Yes	Yes	Cost	Unknown	
	f. Support Software vs. Operational Software	Yes	Yes	Yes	Cost	Unknown	
	3. Form of Cost Estimating Relationship (CER)	Yes	Yes	Yes	Cost	Variable	
	B. Units of Work Method	Yes	Yes	Yes	Cost	Unknown	
	C. Experience Method	Yes	No		Cost	Unknown	
	D. Constraint Method	Yes	No	Yes	Cost	Unknown	
	E. Analogy Method	Yes	No	Yes	Cost	Unknown	
Estimate Analysis Capabilities	F. Percent of Hardware Method	Yes	No	Yes	Cost	Unknown	Can generate extremely large error between estimated and actual cost.
		No	No	Unknown	Cost	Unknown	

Applications:

C/C - Command and Control      BS - Business  
SC - Scientific                      All - All the above  
UT - Utility

an impact on software development costs. However, neither qualitative nor quantitative data was available to investigate the impact of this area on costs.

3.3.2.2 Understanding development tasks. This is an area which has to have an impact on software development costs as well as other elements of life-cycle costs. Communication with the purchaser is an important factor in determining the impact of this area. Since there was only sparse quantitative data with which to measure impacts for this area, the magnitude of the effect of only two factors could be determined.

3.3.2.2.1 Participation of programming organization in requirements analysis. There was quantitative data available to examine the impact of this factor. The degree of participation was stratified into three categories: extensive, intermittent, and minimal. Productivity goes up as participation increases; over the spectrum of participation it increases by 62 percent (between minimal and extensive).

3.3.2.2.2 Misunderstanding requirements. The impact of changes in requirements was examined in paragraph 3.3.1.1.4.2. At other times the developer may misunderstand the requirements as stated by the purchaser and deliver against these misunderstood requirements. This would obviously have an impact requiring software revisions ranging from a minor change to an entire redo of the software package.

3.3.2.2.3 Failure to meet understood requirements. There will be cases where the developer understands the requirements perfectly, yet for a variety of reasons cannot deliver software that meets those requirements with the resources and time allotted. This might be termed the classic overrun. Its magnitude could vary significantly, depending on circumstances. It is generally caused by the other factors discussed in this section.

3.3.2.3 Identifying project tasks. Once the developer assumes responsibility for the development, the effort is usually divided into project tasks. This has traditionally meant dividing the total effort into units which could conveniently be handled by one person. One person may have several units as his responsibility. These units might be identified as modules, programs, subprograms, subroutines, or functions; each one is generally a separately compiled entity. Traditionally, development has started at this level, and then the units are integrated into a working system. This has been called a bottoms-up approach. Under modern programming the reverse is true with development starting at the top. Still, however, work is divided into units that can be conveniently handled by a single person.

3.3.2.3.1 Number of project tasks. If one equates project task with the work unit as described above, there is an extremely strong correlation between man-months for development and the number of work units. A work unit is equated to each separately compiled entity in the development. As with pages of delivered documentation, the number of work units is as good as or a better estimator of the man-months required for development than source statements. However, unlike documentation, it can be used as a predictor at some phase of the development process. An estimate of the number of work units should be available at either the Preliminary Design Review (PDR) or the Critical Design Review (CDR).

3.3.2.4 Design. Design is essentially the responsibility of the developer, although the purchaser may impose certain disciplines such as structured top-down design. Design is an area that has obvious impact on software development costs as well as other life-cycle costs. There is, however, very little quantitative data with which to assess these impacts. A number of design factors were investigated both quantitatively and qualitatively. The interface between requirements analysis and design was discussed previously.

3.3.2.4.1 Complexity. Complexity of design is a factor that is recognized as having enormous cost impact. However, attempts to create a rating scale have not been successful. The rating scale for design complexity selected by SDC for the Phase III Study, [49], which was also unsuccessful, involved four rating levels which ranged from the direct translation of manual tasks to automatic functions, to the automation of undefined and unstructured functions. Analysis of the data indicated that the productivities did not vary in a uniform manner over the rating scale, and that complexity in the presence of other variables did not appear to have an effect on productivity.

3.3.2.4.2 Stability. Stability of design, like stability of requirements, should have a significant development cost impact. SDC Phase III data included a variable to measure this design stability. It could assume four states: design carried through without change, few changes, frequent changes, and almost completely redesigned. It is tacitly assumed that these are design changes occurring with unchanging requirements. Analysis of the data indicated no pattern to the productivities over the rating scale, and little statistical significance in regression. Stability, however, seems to stratify into two groups: (1) no and few design changes falling into a group with high productivities, and (2) frequent design changes and complete redesign falling into a group with low productivities. The former group is about 1.5 times as productive as the latter group.

3.3.2.4.3 Modern programming techniques. The imposition of modern programming as a design method should have a beneficial impact on software development cost. There is no quantitative data to demonstrate this impact, but Aron, [6], estimates a cost savings of 40 percent using modern programming techniques.

3.3.2.4.4 Design techniques. The Central Computer Agency of London, [27], attempted to study the cost impact of various design

techniques such as flow-charting and decision tables. Other than citing the possibility of significant effect, little else was presented about the impact of these techniques on cost. Another design technique that could possibly have an impact on cost is the use of a formal design language. However, there was no data available with which to measure the impact of this technique on cost.

3.3.2.4.5 Flexibility. Design flexibility is usually added so that the software is more easily adaptable to changing requirements. In many cases, design flexibility is added with a set of potential requirements changes in mind, which may not be the requirements that actually change and thus render the design basically inflexible. Based on the data available, design flexibility could not be isolated from requirements changes. Since the requirements for future changes are speculation, adding flexibility may or may not incur a cost benefit. An example is the tradeoff of a data base management system (DBMS) versus a file management system which was described in paragraph 3.3.1.1.3. If volatility in record formats and files is expected, then the additional cost of design flexibility of a DBMS may prove beneficial. On the other hand, if the record formats and files are expected to remain stable, then utilizing a simple file management system is more logical. There is also a tradeoff between design flexibility and testing. Increasing design flexibility will increase the cost of the design phase of software development. Boehm has stated, [117], that 60 percent of the errors discovered in testing require design changes. It stands to reason that if the design is flexible, the error can be corrected with fewer resources. Thus, by adding design flexibility one would expect to increase design costs, but reduce testing costs.

3.3.2.5 Software sizing and cost estimation techniques. Software sizing and cost estimation techniques do not have a direct effect on the cost of developing software, but they do have a large impact on creating

a reliable estimate. One of the major problems is to determine what should be included in software cost estimates for any particular software development.

3.3.2.5.1 Cost per instruction method. The two biggest factors causing unreliability of the estimate using this technique are: the error associated with making the size estimate (number of instructions), and the complete lack of standards regarding the definition of an instruction. A third, but rather minor, factor causing unreliability is whether or not the cost per instruction is to be considered a constant, independent of the number of instructions. This usually manifests itself in being forced to make a choice between a linear Cost Estimating Relationship (CER) (constant cost per instruction) or a geometric CER (variable cost per instruction). The final decision is highly dependent on when the estimate is being made in the development cycle. Table 4 presents estimates of how accurate the sizing parameter may be as a function of phase in the development cycle. The worst case is for new source code "ball park" estimates in concept formulation where the one sigma error of the estimate is approximately 5.0 times the estimate on the upper band and 20 percent of the estimate on the lower band. Estimates with this kind of error are considered unreliable.

3.3.2.5.1.1 Definition of instruction. How the estimator interprets the word "instruction," and the Cost Estimation Relationship (CER) to which that interpretation was applied, has a large impact on the reliability of the cost estimate. Captain Devenny, [40], in his study of ESD cost estimating, gives a couple of examples of how different interpretations of the word "instruction" have created widely different estimates for the same project. In one case it resulted in a difference by a factor of 10. The element causing the largest variance is what software is included in the instruction count. In one ESD example, an estimate included diagnostic and support software, whereas the other did not. For on-board avionics software, an inexperienced estimator may exclude instruction counts for simulation and ATE software, whereas an experienced estimator would include it. Another element causing wide variance is the use of object code

TABLE 4. ACCURACY OF SIZING PARAMETER

ESTIMATOR <sup>a</sup>	DEVELOPMENT STATUS										
	CONCEPT FORMULATION					DEVELOPMENT					
	BALL PARK	BUDGET QUALITY	RFP QUALITY	PDR	CDR	CODING AND CHECKOUT COMPLETE	TEST AND INTE- GRATION COMPLETE				
	-1% +1%	-1% +1%	-1% +1%	-1% +1%	-1% +1%	-1% +1%	-1% +1%				
Total Object Words	67% 200%	60% 150%	50% 100%	24% 30%	17% 20%	9% 10%	0 0				
Total Object Words Minus Data Areas	75% 300%	67% 200%	50% 100%	24% 30%	17% 20%	9% 10%	0 0				
New Object Words Minus Data Areas	78% 350%	75% 300%	67% 200%	43% 75%	17% 20%	9% 10%	0 0				
New Source Code	80% 400%	78% 350%	71% 250%	60% 150%	33% 50%	20% 25%	0 0				

a The probability is 68% that the actual sizes will fall within the stated ranges about the estimate.

instruction counts in CERs developed on the basis of source count and vice versa. Generally speaking, the interchange can only be made reliable if the source code is in assembler, which is close to being a one-to-one relationship. In summary, to get a reliable estimate using the cost per instruction method, the definition on which the instruction count is based must be consistent with the one used in developing the CER.

CER reliability is discussed below for several definitions of the word "instruction." This reliability always assumes that the instruction count is exact for the definition cited (i.e., perfect sizing estimate). Thus, the reliability measures the variance in programmer productivity for the given definition. The definition of the word "instruction," in itself, is not sufficient to create a reliable estimate; therefore, other dimensions of the problem must be taken into account.

3.3.2.5.1.1.1 Total object words. This is the total number of computer words for every program required to run and maintain the system in the field. It would thus include the memory space in the CPU each program allocates for data areas and constants. This count is usually obtained from the output of a linkage editor or binary loader. These programs usually give the internal CPU memory requirements for each module loaded that is required to run the program. If the system uses dynamic storage techniques, which do not appear on the linkage editor output or binary loader map, these requirements must be added. The word count is not normalized by word size. The standard error consistent with this definition of the word "instruction" in ratio form is 182 percent of the estimate on the upper band and 35 percent of the estimate on the lower band. There is a variety of ways a system can be created to occupy a given amount of CPU space, each requiring different resource expenditures.

3.3.2.5.1.1.2 Total object words minus data areas. This is the amount of executable object code in the total object code with all data areas and constants subtracted out. This is usually very difficult

to obtain. Painstakingly, each data area and constant declaration are subtracted from the assembler listing, including the assembler listings which are outputs of HOL compilers. For the average system this amounts to about 13 percent of total object code, but there is a wide variance about this average. An estimate consistent with this definition of the word "instruction" has a standard error of 160 percent of the estimate on the upper band and 38 percent on the lower band.

3.3.2.5.1.1.3 New object words minus data areas. This is the amount of executable object code in the system that resulted from source statements written during the development being considered. This is even more difficult to obtain than the information required to arrive at the parameter in the previous definition. To obtain it, one must take only new modules or ones that had greater than a 50 percent overhaul from previous systems, and apply the same method as used in the definition immediately above. For the average system, this amounts to about 80 percent of the executable object code, but, again, there is a wide variance about this average. Also, one cannot expect this average to remain constant in the future. The percentage has been dropping over the years as more and more canned and reusable software becomes available. One estimate is that by 1990 only 15 percent of a typical delivered system for use by private industry will be new code. The standard error for this definition in ratio form is 140 percent of the estimate in the upper band and 42 percent of the estimate in the lower band.

3.3.2.5.1.1.4 New source lines. This is the number of source statements written during the development that is mapped into new object code in the delivered system. Source statements are only counted for new modules or modules that have greater than a 50 percent overhaul from previous systems. For batch development systems with card input, this count is obtained from a manual count of cards. For terminal oriented systems, in which the source code is prepared with text editors, there is usually a line counter in the editor so the counts can be obtained

automatically. For the reasons discussed in paragraph 3.3.1.1.9 on Language Requirements, the standard error for a source definition of "instruction" decreases considerably from that of an object code definition. For a source code definition the standard error is 68 percent on the upper band and 60 percent on the lower band.

3.3.2.5.1.1.5 Delivered code versus total code. The code delivered to the purchaser, which is required to run and maintain the system, is not always the total amount of code created by the developer during the development. The developer may have created additional software such as simulations and other test tools to check the software, which are not delivered to the purchaser. Since the amount of this kind of software varies among developments it creates variance in an estimator based on delivered code. For the SDC Phase III data, delivered code as a percent of total code averaged 77 percent with a standard deviation of 30 percent.

3.3.2.5.1.1.6 Support software versus operational software. The definitions for "instruction" were concerned with the software required to run and maintain the system, which may exclude certain classes of support software used to develop the operational software. Depending on the development, various amounts of support software are created, which can cause error in an estimator that excludes it. Developments with a new CPU or new languages generally have a large amount of this type of software, a good portion of which is never delivered to the customer. Thus, a considerable amount of this kind of software falls into the category of non-delivered code. The average percent of total code for the typical system is not known, neither is the variance, but it is considered to be large. The percent of support software that is delivered in the future is expected to increase, especially for federal government programs. Federal agencies and particularly military departments are demanding that support software be delivered with the operational software, since in most cases they financed its development.

3.3.2.5.1.2 Form of Cost Estimating Relationship (CER). The variance in the estimate caused by the form of the Cost Estimating Relationship (CER) chosen is small compared to that caused by the sizing estimate and definition of "instruction." The forms used are linear (constant cost per instruction) or geometric (cost per instruction varies with program size). A number of investigators have proclaimed that the cost per instruction should not remain constant (independent of the size of the effort). With larger projects more personnel interactions have to occur, resulting in decreased personnel productivity. Yet, other sources have indicated that economics of scale may be effected, [59]. The generalized forms of the relationships are as follows:

$$MM = aI^b \quad \text{(geometric)} \quad \text{(Equation 1)}$$

$$MM = c+dI \quad \text{(linear)} \quad \text{(Equation 2)}$$

where

MM = man-months for development; I = number of instructions; and a, b, c, d = constants

A number of investigators have come to the conclusion that b approximates 1.25. This study investigated both forms, and for some sub-populations of programs the linear form was a better estimator. However, for the total population and major elements of the population (command and control, scientific, utility and business programs), the power function is recommended, and b was found to range from 0.72 to 1.26, depending on the application.

3.3.2.5.2 Units of work method. This method, described in paragraph 3.3.2.3, involves structuring the total software development into work packages or modules, that can be conveniently handled by one person. One person, however, may be responsible for several modules. Each module is costed separately, and then the individual costs are summed to arrive at a total project cost. The modular structure of a program is often not

available with any accuracy until well into the development (at the completion of design). Usually, by this time 40 percent of resources will have been expended. The variance associated with this method is unknown. However, analysis of the available quantitative data indicated it is superior to the cost per instruction method. Thus, when the sizes of the work packages are known, the cost per instruction method is applied to the individual packages rather than to the total program. This more refined level of analysis will result in more accurate estimates. Estimating the sizes of small packages can be done more accurately, and by summing the sizes of the small packages, the over- and underestimates of the small packages will also tend to offset each other, resulting in greater accuracy in the estimate of overall size. To assess the remaining expenditures anticipated, the actual costs experienced to date will be subtracted from the total estimate obtained by summing the estimated costs of the work packages. Aron, [7], states that this method can only be applied successfully to small projects, and that it is the worst of all possible methods for large projects. His contention is based on the assumption of estimating 1.5 man-months of effort for every module in the system.

3.3.2.5.3 Experience method. This method relies on the experience of the personnel involved in the development. It is based on the premise that experience on a large programming system can be carried forward to similar systems. This method tends to lose its effectiveness if it is applied to systems which are either larger or functionally different than those reflected in the experience of the project personnel. Aron, [7], states that it is the best of all possible methods when it can be applied. There is no quantitative data as to the variance associated with this method. Although the method may be viable for a developer, it is not a viable method for the purchaser for it can be interpreted as "trusting the cost estimate of the contractor if he has developed similar programs."

3.3.2.5.4 Constraint method. This is the software version of design-to-cost. Based on schedule, dollar, or manpower constraints, the developer simply agrees to do the job within the constraints. The problem is

to come up with a set of satisfactory specifications that can be met within the constraints. This requires close liaison between the purchaser and developer before project initiation. An overrun occurs when the developer has used up the resources within the constraints, and has not met the agreed upon specifications. The variance associated with this method is not known. The close liaison between the purchaser and developer, usually required by this method, is thought to be beneficial, but Aron, [7], rates the method third in accuracy behind experience and cost per instruction.

3.3.2.5.5 Analogy method. This is similar to the experience method but on a more aggregate scale. In this method the project manager selects a software system which is most closely analogous to the one to be developed, and simply states that the cost of the new software system will be the same in constant dollars. The variance associated with this method is not known. This method probably tends to overestimate when the new system is quite similar to the analogous system, because it does not take into account the effect of prior learning and reused software. DAI has found an offshoot of this method to be quite effective in estimating program size for the cost per instruction method. Analogy can be very effective in estimating the total object word requirements for a proposed system. How those object words are to be generated is, however, a major cost variant. Thus, analogy is no better than using the total object word definition in the cost per instruction method.

3.3.2.5.6 Percent of hardware method. In this method the RDT&E hardware cost is multiplied by some factor to arrive at the software cost. The tacit assumptions in this method are:

- (1) one can estimate RDT&E hardware costs better than independently estimating software costs, and
- (2) software costs tend to be a somewhat constant percentage of hardware costs.

The first assumption is probably valid in most cases, but the second is suspect. Boehm, [16], has estimated hardware/software trends for Air Force ADP systems. His results are reproduced in Figure 7. Boehm stated no variance about the trendline. Using the results of the curve in the current time frame, software is approximately 67 percent of total system cost; thus RDT&E hardware cost would be multiplied by 2 to arrive at software cost. It is the experience of DAI that there is a wide variance about this hardware/software ratio. It is not unusual for software to be as high as 80 percent or as low as 50 percent of the total system cost. Using this method, there could be errors of over 100 percent in the software cost estimate, even if the hardware estimate were exact. This method rates as the lowest in accuracy of all those discussed.

3.3.2.6 Estimate analysis capabilities. The ability of the developer to conduct in-depth analysis to estimate the cost of software development projects should have an impact on software costs; however, neither qualitative nor quantitative information was available to investigate the impact of factors from this area.

3.3.3 Developer domain--project management procedures. Table 5 lists the management procedures analyzed and summarizes the effects of the factors in the developer domain.

3.3.3.1 Cost estimation assumptions. There are a number of assumptions the developer can make that can create variance between the estimate and the actual cost of delivering the code. The variances discussed in paragraph 3.3.2.5 on software sizing and cost estimation techniques are also directly applicable to the area of cost estimation assumptions.

3.3.3.1.1 Size. How large the developer estimates the effort to be, based on purchaser requirements, can cause a large variance between estimated and actual cost. If the developer sizes on the basis of instructions,

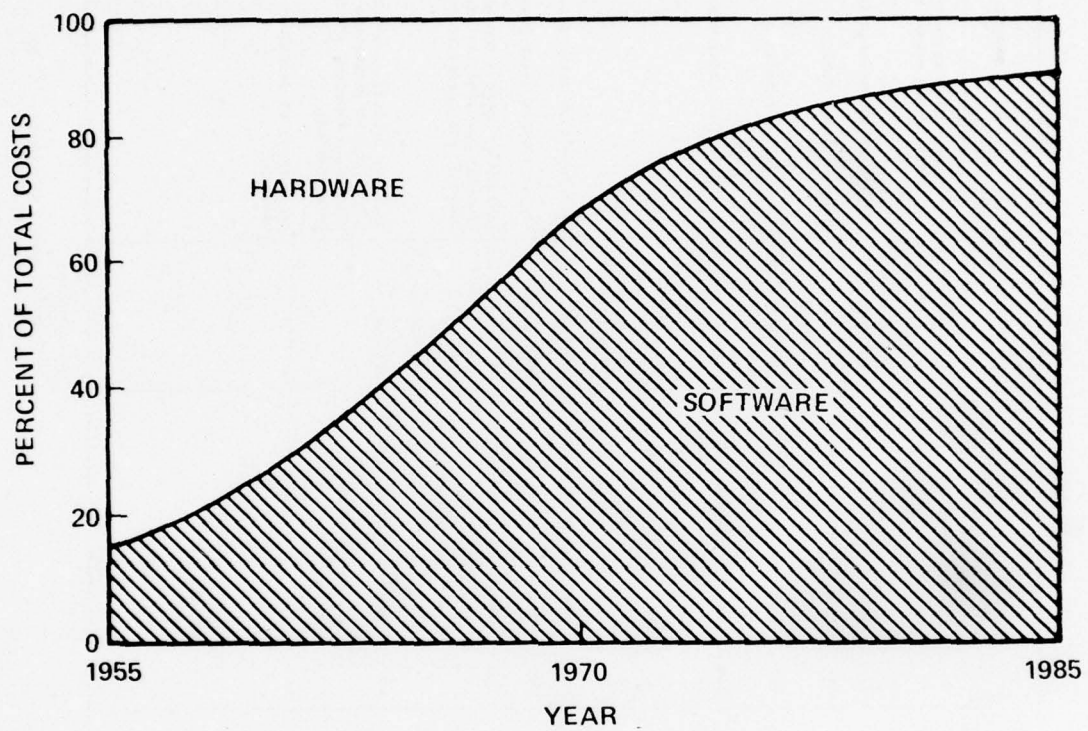


Figure 7. Hardware/software cost trends  
(Boehm [16])

TABLE 5. DEVELOPER DOMAIN PROJECT MANAGEMENT PROCEDURES

Area	Factor Investigated	Type of data analyzed Qual. Quant.	Impact	Dependent variable affected	Effect	Comments
Cost Estimation Assumptions	A. Size	Yes	No	Yes	Variance of Actual to Estimated	Major effect Can cause major variance between contractor estimate and actual cost.
	B. Personnel Productivity	Yes	No	Yes	Productivity	Major effect Another major variance factor.
	C. Direct Application of Previous Experience	Yes	No	Yes	Productivity	Major effect Major variance factor.
	D. Secondary Resources	Yes	Yes	Yes	Cost	Minor variance factor.
Management Experience and Competence	A. Experience with Language	Yes	Yes	Yes	Productivity	Decrease with experience less than 6 months with language No effect with greater than 6 months experience
	B. Experience with Application	Yes	Yes	Yes	Productivity	Increase No correlation using quantitative data.
	C. Overall Experience	Yes	Yes	Yes	Productivity	Increase No correlation using quantitative data.
	D. Personnel Mix by Type	Yes	Yes	Yes	Productivity	25% decrease for each 10% increase in support personnel Mixture of support personnel to programmers and analysts can have an impact. However, mix cannot be dictated.
Requirements for Collection of Data for Estimate Analysis	A. Work Breakdown Structure (WBS)	Yes	No	Yes	Variance of Actual to Estimated	Variable Variance depends on consistency of definitions between estimation methods and what costs requirements dictate are to be collected.
	B. Cost/Schedule Control Systems Criteria (C/SCSC)	Yes	No	Yes	Variance of Actual to Estimated	Variable Consistency of definitions again affects variance.

## Applications:

C/C - Command and Control      BS - Business  
SC - Scientific                      All - All the above  
UT - Utility

TABLE 5. DEVELOPER DOMAIN PROJECT MANAGEMENT PROCEDURES (Continued)

Area	Factor Investigated	Type of data analyzed Qual. Quant.	Impact	Independent variable affected	Effect	Comments
Cost Collection Practices	A. Amount and Method of Collection	Yes	No	Yes	Cost	Unknown
	B. Frequency of Collection	Yes	No	Yes	Cost	Unknown
	C. Information retrieval and Analysis of Collected Data	Yes	No	Yes	Cost	Unknown
						Collection requires resources, but it may be recouped in a smoother running project. Automating collection could reduce cost impact.
						Minor impact.
						Magnitude of effect would depend on size of effort.

Applications:

C/C - Command and Control  
SC - Scientific  
UT - Utility  
ES - Business  
All - All the above

the error can be appreciable, especially for programs in which the exponent of the estimating relationship is greater than 1.0. If the developer sizes directly in man-months and secondary resources based on previous experience, then the variance essentially depends on the applicability of his previous experience to the new development.

3.3.3.1.2 Personnel Productivity. If the developer sizes the software in terms of a product measure such as the number of instructions or number of modules, then his assumed personnel productivity against those measures is a key variant in the estimate. Since producing software is a very labor intensive activity (greater than 85 percent of resource expenditures), assumed personnel productivity is the key to arriving at an accurate cost estimate. Since generating software is creative, a wide variance in personnel productivity can be expected. This is especially true when the product measure is number of instructions. If the developer knows his personnel and the requirements of the project well, his estimate should be better than that of the purchaser which is based upon the productivity experience of several developers with various types and sizes of software developments.

3.3.3.1.3 Direct application of previous experience. The amount of previous experience that can be applied to the new development is another factor that can create variance between estimated and actual expenditures. If a developer has no applicable experience, then his estimate will be in error by the amount of benefit that could be attributed to experience. The benefit of experience can be manifested in several ways. As examples, the transport of software from an operational system to the development system, more accurate estimates of size, and greater productivity are a few of the benefits. As discussed in paragraph 3.3.1.1.14, the transportability of software can prove easy or difficult depending on the situation. If a large amount of software can be transported easily, it can make a large difference in the cost of the system.

3.3.3.1.4 Secondary resources. Since secondary resources are small compared to the primary resource (man-months), any variance associated with their estimate will generally only create a small error in the overall estimate. This is true especially if the estimate of the primary resource is used to estimate the secondary resources. The main secondary resource to be estimated is machine utilization. Another resource considered important is documentation production. Analysis of the available quantitative data indicates that both machine utilization and documentation are highly correlated with the primary resource; they are estimated as approximately 7.5 percent of the cost of the primary resource.

3.3.3.2 Management experience and competence. Management experience and competence is not only a factor in software development, but as discussed previously, it can be a factor in the cost estimate generated by the developer. Factors within this area are of major importance in determining software development costs, but it is extremely difficult to demonstrate this with the available quantitative data.

3.3.3.2.1 Experience with language. The experience of the developer with the programming languages being used for development, measured in average number of years for the assigned staff, has virtually no correlation with the productivity that will be achieved. Halstead, [61], states that it only takes six months for programmers to become proficient in a new language. Therefore, the contention is that if language is an element in the development, expect as much as six months of lower than normal productive effort on the part of the programmers involved.

3.3.3.2.2 Experience with application. Although an increase in productivity would be expected, the analysis of quantitative data indicated no correlation with productivity.

3.3.3.2.3 Overall experience. Combining the experiences with language and application did not improve the correlation with productivity.

In addition, Kosy, [82], cites a controlled laboratory study by Sackman which comes to similar conclusions. Two different problems were given to a group of 12 programmers. The resulting productivities did not correlate with programmer experience. Inherent programmer abilities completely dominated the effects that could be attributed to any cause.

3.3.3.2.4 Personnel mix by type. The mix of personnel on a project can have an impact on productivity. Malone, [86], studied 11 software developments in which the personnel were classified into three categories: programmers, analysts, and support personnel (management, clerical, etc.). He determined that productivity dropped about 25 percent for each 10 percent increase in the amount of support personnel. The percentage of support personnel, however, is not something that can be dictated by fiat. The expected mix for the normal project is about 20 percent support personnel and 80 percent programmers and analysts.

3.3.3.3 Requirements for collection of data for estimate analysis. These requirements dictate the software cost and related data that will be collected. Since there are no generally accepted standards defining software cost data, there can be considerable variance between estimated and actual costs.

3.3.3.3.1 Work Breakdown Structure (WBS). Any large system development for the Department of Defense will have a Work Breakdown Structure (WBS) against which costs can be collected and monitored. The way in which software appears in the WBS can have a significant impact on the "apparent" cost of software for the development under consideration. The WBS of a major hardware systems development may contain very little of the associated software development WBS, even though a considerable amount of the total systems development cost is for software development. In some cases, the systems WBS has contained only one element for software. This element usually reflects only the coding and debugging costs of applications software. If the cost estimate for software development included the analysis, design and test of the applications and support software, the estimate will be considerably greater than the reported costs.

If the project is primarily a software development effort, a highly refined WBS should be a standard procedure. Programs on which cost reporting and control is effected at a refined level of WBS indenture have had greater success at cost control.

3.3.3.3.2 Cost/Schedule Control Systems Criteria (C/SCSC). DoD instruction 7000.10 prescribes a format for reporting costs which is required for large weapon systems developments. The reports received under the system are called CPRs (Cost Performance Reports). Software can be reported in two different formats: the functional format and the WBS format. A major variance between the estimated and actual costs can occur if there are inconsistencies between the definitions of software used to estimate and to report costs.

3.3.3.4 Cost collection practices. Cost collection practices by the developer can have an impact on software development cost if data is to be collected. In some cases the purchaser may also have to expend resources to collect the data. This may occur, for example, if the purchaser has personnel at the development site to monitor the program (for schedule and cost control). These additional costs may, however, be recouped through improved management of the program. The degree of automation in data collection also has an impact on cost; in addition to cost and performance data, product measurement parameters, such as instruction counts, can be collected. There was no quantitative data available on the cost impact of cost collection practices; however, some of the more important aspects of cost collection which impact development cost are discussed.

3.3.3.4.1 Amount and method of collection. The amount of data collected is essentially a function of the WBS as dictated by the cost data collection requirements. The more data that is collected, the greater the software development costs. The costs may, however, be recouped if it provides improved management tools with which to control costs.

The degree of automation in the collection process can have a significant effect on the cost impact of cost data collection. Substantially fewer man-hours are required to record and report data automatically. However, if a major development effort is required to automate the cost data collection procedures, an adverse impact on the program costs could result. A consideration in the evaluation of automation costs is the use to which the automatic aid can be made in subsequent development efforts.

3.3.3.4.2 Frequency of collection. For most DoD efforts, cost data is collected and reported monthly. Therefore, this variable will only have an effect if collection is on other than a once a month basis.

3.3.3.4.3 Information retrieval and analysis of collected data. For some projects the amount of cost data reported monthly is enormous. Therefore, it creates an information retrieval problem for the purchaser which can generate extra cost expenditures. For example, cost reporting for the Army Patriot Missile under CSCS generates a monthly report of computer printouts approximately 4 inches thick. Not all of this, of course, relates to software cost. For certain elements it reports down to the seventh level of WBS indenture. It also comes in the form of a magnetic tape, so that all data is in machine readable form. The sheer volume of a report of this nature can create a large information retrieval problem for the developer, who must collect and generate the data, and the purchaser, who must analyze the data. Generating data across reports to establish trends is even more difficult. For the Patriot Missile, a computerized information retrieval system is being developed to aid in this analysis procedure.

3.3.4 Developer domain--determining actual costs. This category of variables is probably responsible for generating the greater portion of variance between estimated and actual cost. The WBS dictates what is reported as actual software development cost. The variance between this

reported cost and that estimated can be large if the cost definitions on which the estimate is based are not consistent with the software WBS. The factors studied are outlined in Table 6.

3.3.4.1 Personnel charges. The allocation of personnel charges on a project can have a large impact on reported actual costs. In some developments, for example, management and other support personnel may not be charged against software, even though they are performing software development functions. Their labor costs may be charged against Systems Engineering and Project Management in the Work Breakdown Structure (WBS). In other developments the same type of personnel performing the same functions may be charged against a software element in the WBS. As an example, in a primarily hardware development organization, the development of algorithms to be implemented in software may not have the design engineering time charged against a software element in the WBS. Conversely, in a primarily software development organization the opposite would be expected. At one extreme, personnel charges against software may only include programmer time during code and debug, while at the other extreme, the charges can also include programmer, management, and secondary support time through all phases of the development.

3.3.4.2 Development phases included. The software development phases included in software cost can have significant impact on the variance between actual and estimated cost. In a number of cases, software development cost reflect the cost for code and debug only and disregard the other phases of development. On the average, code and debug only amount to 20 percent of the total development costs (if analysis and design, and testing and integration are included in the total).

3.3.4.3 Factoring hardware from software costs. In large weapons systems developments, software costs are often not detailed in the Work Breakdown Structure (WBS) even though software costs may be a large portion of the expenditures. In such cases, there may not be a requirement

TABLE 6. DEVELOPER DOMAIN DETERMINING ACTUAL COSTS

Area	Factor Investigated	Type of data analyzed Qual. Quant.	Req. act.	Dependent variable affected	Effect	Comments
Determining Actual Costs	A. Personnel Charges	Yes No	Yes	Cost	Variable	Some developments may only charge programmer time against software, others may include other support personnel.
	B. Development Phases Included	Yes No	Yes	Cost	Variable	Some developments may only charge code and checkout against software, others may include in addition, analysis and design and testing and integration.
	C. Factoring Hardware from Software Costs	Yes No	Yes	Cost	Variable	Some developments have a large degree of hardware/software bundling.
	D. Separating Development from Maintenance Costs	Yes No	Yes	Cost	Variable	The large amount of development that sometimes occurs after the system goes into the field makes it difficult to separate maintenance from development costs.
	E. Organizations Included	Yes No	Yes	Cost	Variable	Usually only developer costs are included from development to development purchaser may assume different responsibilities causing variance between estimated and actual costs.
	F. Secondary Resources	Yes No	Yes	Cost	Variable	Since secondary resources generally amount to less than 7.5% of primary resources, any error induced by how they are charged will be minor.

Applications:

C/C - Command and Control  
SC - Scientific  
UT - Utility  
ES - Business  
All - All the above

to make a software cost estimate, but the presence of software can create a large variance between estimated and actual total systems cost. An approach for estimating software costs in such a case would be the Percent of Hardware Method discussed previously.

In some system developments, personnel may be dually qualified in both hardware and software. This is especially true for engineers in the field of avionics development. The method used in charging their time is critical to the software costs generated. When certain kinds of engineers are qualified to perform the entire spectrum of work effort from hardware through firmware (which originates as software) to software, it is very difficult to ensure that the time is allocated consistently (especially among projects and among development organizations). Also, in certain kinds of development functions, such as total systems integration and test, it is also very difficult to distinguish between hardware and software activities. Thus, differences existing in the allocation criteria for functions involving hardware and software can significantly affect reported costs.

3.3.4.4 Separating development from maintenance costs. For a large number of systems it is difficult to determine where development ends and maintenance support commences. After the system is installed there still may be development activities, which are often referred to as enhancements. How these costs are charged can make a considerable difference between estimated and actual software development costs.

3.3.4.5 Organizations included. In most large developments there are resources expended on the part of the purchaser as well as the developer. Costs, however, are usually only allocated against the developer. The ratio of purchaser cost to developer cost cannot be assumed to be relatively constant over a variety of different developments. In some developments the purchaser will assume responsibilities that the developer

assumes in others. If expenditures are not included from both organizations, considerable variance between estimated and actual costs can occur among development programs.

In some developments more than one kind of organization can be involved in the development. For example, the final system may be developed through the combined efforts of military, civil service, and contractor personnel. Within such a development there may also be a number of different project offices and a number of different independent contractors involved.

3.3.4.6 Secondary resources. Paragraph 3.3.3.1.4 mentioned the high correlation between primary and secondary resource usage in software development. How these secondary resources are charged in the Work Breakdown Structure (WBS), however, can make a considerable difference in the actual reported cost associated with them. They may be charged directly against software WBS elements, against non-software WBS elements, be included in G&A, or some combination of the three.

3.3.5 Other factors--type of application. There are some factors which affect cost that do not fall conveniently in either the purchaser or developer domain. Some of the factors that were discussed previously overlap both domains, but were placed in the domain having major responsibility. The only factors that appear outside of both domains are the types of software application. For purposes of discussion, the types of applications have been divided into the following: business, scientific, utility, command and control, and on-board avionics. (See Table 7.)

3.3.5.1 Business. Business applications on a cost per source line basis are usually more productive than non-business applications, since business applications are normally much less complicated than other applications, such as scientific or command and control. Analysis of available

TABLE 7. TYPE OF APPLICATION

Area	Factor Investigated	Type of data analyzed		Impact	Dependent variable affected	Effect	Comments
		Qual.	Quant.				
General	A. Business	Yes	Yes	Yes	Productivity	Increase	Magnitude of effect a function of program size.
	B. Scientific	Yes	Yes	Yes	Productivity	Decrease	Magnitude of effect a function of program size.
	C. Utility	Yes	Yes	Yes	Productivity	Increase	Magnitude of effect a function of program size.
	A. Command and Control	Yes	Yes	Yes	Productivity	Decrease	Magnitude of effect a function of program size.
	B. On-Board Avionics	Yes	No	Yes	Productivity	Decrease	Least productive
	1. On-Board Flight Programs (OFF)	Yes	No	Yes	Productivity	Decrease	About equal to Scientific.
Air Force Systems Command	2. Simulation	Yes	No	Yes	Productivity	Increase	Most Productive.
	3. Automatic Test Equipment (ATE)	Yes	No	Yes	Productivity	Increase	Most Productive.

Applications:

C/C - Command and Control  
 SC - Scientific  
 UT - Utility

BS - Business

All - All the above

quantitative data indicates that productivity should increase over that of the total population.

3.3.5.2 Scientific. The quantitative data analyzed indicated that software development for scientific applications was less productive than that of the total population and the business and utility programs. It was also noted that software development in scientific applications was somewhat more productive than those of command and control.

3.3.5.3 Utility. Analysis of available quantitative data regarding the software development of utility programs, such as I/O drivers, etc., indicates that productivity is greater than that of the total population. Productivity in utility programs, considered a part of support software was lower than that of business programs, but higher than that of scientific, and command and control programs.

3.3.5.4 Command and control. There are a number of diverse functions of software involved in command and control applications such as data base management, information retrieval, and display generation. However, available qualitative data was not adequate to analyze the command and control functions separately. Analysis of overall command and control applications indicated that productivity was less, as compared to the other types of applications. Since command and control software is often larger, lower productivity would be expected.

3.3.5.5 On-board avionics. Unlike command and control, on-board avionics software conveniently subdivides itself into three different types: On-board Flight Program (OFF), simulation, and Automatic Test Equipment (ATE). Accordingly, one can utilize different Cost Estimating Relationships (CERs) for each of the three different applications of software. OFFs usually have to operate in real-time, quick-response, memory constrained environments, which naturally have an impact of lowering development productivity. In addition, OFFs usually require a great amount of testing

inasmuch as a number of them are life critical, which also has an impact of decreasing productivity as compared to other types of applications. Analysis of qualitative data indicated that development of simulation software could be expected to be about as productive as normal scientific software. Developing ATE software should be the most productive of the three, since it is usually developed with very little testing.

#### 4. TECHNIQUES FOR IMPROVING RELIABILITY OF SOFTWARE COST ESTIMATION

##### 4.1 Criteria for selection of techniques

The identification and analysis of factors that influence software cost estimation served as the basis for selecting techniques that should be employed to improve such estimates. The factors causing the errors in the estimates encompass hardware system design and performance requirements, software design and performance requirements, descriptors of the environment in which the software is developed, characteristics of the purchaser, and characteristics of the developer. The effect of controls are often constrained by options available to the software purchaser or developer. Consequently, several controls to an effect are frequently offered and tradeoff analyses are implied.

To identify the techniques for improving the reliability of the cost estimates, the factors were divided into two categories reflecting the extent of probable impact on software cost estimations. Factors causing significant impacts were designated as Primary Factors, listed in Table 8, and those factors causing negligible impacts were designated as secondary factors. Techniques for controlling the primary factors have been proposed; they evolved from DAI Corporate experience and literature. They were evaluated in terms of their ease of implementation. Controls of the secondary factors were not proposed since the benefit expected to be accrued from their employment would probably not be worth the implementation effort.

##### 4.2 Employment of techniques

Since the factors affecting the cost estimates exist for the most part in the developer and purchaser domains, the techniques proposed for improving the reliability of software cost estimating are applicable to both domains. Benefits will be accrued to the developer and purchaser

AD-A042 264

DOTY ASSOCIATES INC ROCKVILLE MD

F/G 9/2

SOFTWARE COST ESTIMATION STUDY. VOLUME I. STUDY RESULTS.(U)

JUN 77 J H HERD, J N POSTAK, W E RUSSELL

F30602-76-C-0182

UNCLASSIFIED

TR-151

RADC-TR-77-220-VOL-1

NL

2 of 4  
ADA042264



TABLE 8. FACTORS AFFECTING COST ESTIMATION ACCURACY  
FOR WHICH CONTROLS ARE PROPOSED

PURCHASER DOMAIN

- Performance Specifications
  - Special displays
  - Data management
  - Definition of operational requirements
  - Changes in operational requirements
  - Interface to design
  - Response time requirements
  - Time (hardware) constraint
  - Memory (hardware) constraint
  - Time and memory constraint
  - First development on computer (CPU)
  - Concurrent development of hardware
  - Time CPU specified in schedule
  - Requirement for innovation
  - Language requirements
  - Quality requirements
  - Reliability requirements
  - Testing requirements
  - Transportability requirements
  - Maintenance requirements
- Program Management
  - Development schedule
  - Communications
- Development Environment
  - Developer using another activity's computer
  - Programmer access to computer
  - Operational site development
  - Development and target computer different
  - Number of development locations
  - Programming environment
  - Support software
  - Programming facilities
- Production Environment
  - Multiple software utilization sites
- User Interfaces and Participation
  - End user requirements

TABLE 8. FACTORS AFFECTING COST ESTIMATION ACCURACY  
FOR WHICH CONTROLS ARE PROPOSED (Continued)

DEVELOPER DOMAIN

- Design
  - Complexity
  - Stability
  - Modern programming techniques
- Estimation Techniques
  - Sizing estimate errors
  - Definition of instruction
- Experience and Competence
  - Personnel mix
- Data Collection
  - Work Breakdown Structure (WBS)
  - Cost/Schedule Control System Criteria (C/SCSC)
  - Amount and method of data collection
- Secondary Resources
- Applications

by more accurate estimates of software cost. It is unlikely that the data regarding many of the primary factors will be available to the cost estimator for initial estimates in the conceptual phase. Consequently, less accuracy in the estimate is expected at this time. However, as the data becomes progressively more available during the conceptual and subsequent phases of development, the accuracy of the cost estimates should improve. And, increasing controls can be effected to ensure the estimates remain accurate.

#### 4.3 Control of the primary factors

For each primary factor having an impact on the accuracy of cost estimation, methods are discussed which offer alternative actions for improving the accuracy of the estimates.

4.3.1 Special displays. The measures implemented to control the effect on cost is related to the magnitude of the effect. The alternatives to be considered are:

- If the software requirements for special displays are less than 10 percent of the overall software requirements, the effect on costs is considered negligible and should be accepted.
- If the software requirements for special displays are greater than 10 percent, consider utilization of existing displays (with currently available supporting software) to reduce the requirements for special displays.
- Consider possible reduction and/or elimination in display complexity if feasible without seriously degrading overall operational performance.
- If none of above can be effected, accept the increased costs.

4.3.2 Data management. Several tradeoffs should be considered when considering file management techniques as opposed to Data Base Management System (DBMS) techniques. A file management system will reduce development cost, but increase maintenance costs. A DBMS will increase development cost, but decrease maintenance costs. Among the considerations to be made include the following:

- If there is not expected to be much volatility in the types and format of data to be handled over the life of the system, then opt for a File Management System. If great volatility is expected, then opt for a DBMS.
- Expect to accept greater inefficiency in both the CPU time and memory domains when using a DBMS.
- If the choice is left to the developer, the project manager should be aware of the tradeoff between development and maintenance costs.
- This is not a factor of importance for avionics applications, since any data management that will be required can usually be handled easily by a File Management System.
- For Command Control applications this factor can be important, especially for systems with large data management tasks.

4.3.3 Definition of operational requirements. The delineation of operational requirements is requisite to accurate cost estimates. The level to which the requirements are defined will determine the magnitude of the error. The following guidance is offered:

- Determine what procedures can be implemented in requirements analysis to assure adequate detail. There are no hard and fast rules available, but the project manager should keep requirements analysis high on his awareness scale. Do not let design start until both the purchaser and developer feel comfortable with the requirements. As a minimum:
  - Identify and delineate all software functions
  - Define all operational characteristics and constraints
  - Develop all the software algorithms
  - Perform hardware and software tradeoffs
  - Develop functional alternatives to Computer Program Configuration Items (CPCIs)
  - Identify language requirements
  - Identify hardware constraints
- Consider the utilization of a formal requirements analysis language for the application area involved. There are, for example, machine resident languages available for application areas like Ballistic Missile Defense. The output of such a language is input to design, again usually a formal language. If such a language is available or could be modified, it can be used to assure that a sufficient level of detail is attained in requirements analysis. However, the use of the language will not automatically guarantee the level of detail required.
- Be extremely careful how requirements analysis is paid for, especially algorithm development. In many instances it is done by the purchaser, and therefore may not appear as a software development cost. In some instances it is done by the developer, but is not delineated as a software cost. In other instances it is done by the developer, and is delineated as a software cost. Combinations of the above also occur. These considerations should be taken into account when structuring the Work Breakdown Structure (WBS) and making cost estimates.

4.3.4 Changes in operational requirements. Experience has shown that changes to operational requirements can have a profound impact on the software development costs.

The problem occurs more frequently on large programs; it is unlikely that a large program will be developed without such changes. Consequently, provision for changes and their effects should be included in program plans, and should be accomplished through ECPs or other documentation, depending on the size or scope of the change.

A requirements change will often result in the discard of some existing code and the addition of new code. Sometimes, however, only the latter is involved. In either case, the cost should be expected to increase and the schedule expected to slip due to the requirement changes. Each change should be addressed independently in terms of the amount of code to be written to accommodate the change. (For cost estimating purposes keep track of code discarded.) The amount of code to be written for the change can then be translated into a cost and schedule impact.

Thus, the manager for software development should effect the following:

- Identify areas in which changes are likely to occur
- Perform tradeoff analyses among the change options
- Develop plans to effect potential changes
- Evaluate impact of options on cost

4.3.5 Interface to design. A very detailed requirements analysis can be carried out, but if the outputs of the analysis do not interface clearly to design (i.e., serve as useful input) the effort is essentially wasted. This difficulty can usually be overcome by using one of the two following alternatives:

- If available, impose a formal machine resident requirements analysis language and companion design language on the development. The advantages and disadvantages of this alternative are covered under the Definition of Operational Requirements.
- Have a representative or representatives of the intended programming organization participate in the requirements analysis, along with end user representatives. If this is not possible, have someone with extensive programming experience participate in requirements analysis.

4.3.6 Response time requirements. Software responding in real time is significantly more expensive than that with slower response times. It is very important to on-board flight programs in avionics software developments, less critical to command and control applications, and almost non-existent in business applications.

Factor the software into response time domains, at least to the point of getting the real-time dependent portion isolated. Use the following guidelines once the total software development has been partitioned relative to response time:

- If the real-time dependent portion represents less than 10 percent of the total development, then no special provisions are necessary. Just expect lower programmer productivity for that portion of the development.
- If the real-time dependent portion represents more than 10 percent of the total development, then the following alternatives should be considered:
  - Consider a hardware tradeoff to minimize costs while satisfying performance requirements. If only one system is being developed, the additional cost of hardware will probably be less than the high cost of software (caused mainly by rewrites) to satisfy the response time requirement. If more than one system is being installed in the field, attempt to determine the break-even point in number of systems. If the projected number to be installed is less than the break-even point, consider the hardware alternative. If the projected number to be installed is greater than the break-even point, consider leaving the intended task in software.
  - Consider a firmware tradeoff. Software still has to be developed for this alternative since all firmware originates as software. Software targeted for firmware is generally referred to as microcode. Writing microcode to meet a response time requirement will be less productive per line than ordinary software, but will not generally require the number of rewrites that meeting a tight response time requirement in ordinary software would require. Depositing critical functions in firmware will relax CPU time loading. The firmware option will require an additional hardware cost for each system, since a high speed memory will be required for the microcode to reside in. However, this increased hardware cost per system will not be as large as the total

hardware option examined above. For a single installation development, depositing critical response time functions in firmware will probably pay off. For multi-installation developments, use the guidelines presented above for the hardware option.

- Examine the use of a faster more powerful CPU. This will increase hardware costs, and also impinge on weight and volume constraints, if applicable. The guidelines presented in the hardware tradeoff above apply.
- Consider a multiple CPU system. This is a possibility if several real-time dependent functions are vying for CPU service simultaneously in an interrupt driven system. The CPU time loading can sometimes be relieved by spreading the functions across multiple CPUs. If, however, one function is the driving factor, then this is not a viable alternative. This alternative will increase hardware costs, and also impinge on weight and volume constraints, if applicable. The guidelines presented in the hardware tradeoff above apply.
- Obtain relaxation and/or removal of real-time requirements (obtain and document user concurrence).
- If required, accept a reduction in programmer productivity with concomitant increase in software costs.

4.3.7 Time (hardware) constraint. If the CPU is projected to operate in a time constrained mode, the cost is expected to increase appreciably. A time constrained mode is defined as more than 65 percent utilization of available CPU time for the most demanding task. When the utilization exceeds 80 percent the constraint is considered critical.

This factor is highly correlated with the response time factor, since the constraint is most often present in real-time environments. However, not all real-time environments will present a constraint. For example, a minicomputer controlling machine tools will be in a real-time environment, but the time-constraints are not severe. In contrast, navigation, fire-control, and signal processing computers in an avionics system will most definitely be affected by the constraint.

When a time constraint exists, divide the software into time constrained and non-time constrained tasks using the 65 and 80 percent CPU loading criteria mentioned above. Use the following guidelines after the software has been partitioned in this manner:

- If the time constrained portion represents less than 10 percent of the total development, then no special provisions are necessary. Just expect lower programmer productivity for that portion of the development.
- If the time constrained portion represents more than 10 percent of the total development, then use the alternatives presented for the response time factor.

4.3.8 Memory (hardware) constraint. When software development is constrained by the size of the processor program memory, costs are expected to increase over a similar development without a memory constraint.

- Determine size requirements of the program memory.
  - If the estimate of requirements is less than 60 percent of total memory, assume little or no effect.
  - If the estimate is greater than 60 percent and less than 80 percent of total memory, anticipate increased costs of 30 percent.
  - If the estimate is greater than 80 percent, assume major impact on costs (increases of as much as 200 percent can occur).
- If memory utilization is greater than 60 percent,
  - consider hardware and firmware tradeoffs as discussed under the response time factor. The least expensive hardware alternative, which was not discussed under response time, is to add memory to the proposed CPU. If, however, the proposed CPU is fully configured with memory, then this will not be a viable alternative;
  - consider relaxation of operational requirements to decrease memory requirements (user concurrence should be requested and documented);
  - if required, accept a reduction in programmer productivity, for the necessary extra effort required to make the software fit, with concomitant increase in software cost.

4.3.9 Time and memory (hardware) constraint. If the software development is constrained in both time and memory domains of the CPU, use the guidelines presented under the time and memory constraints factors treated individually (paragraphs 4.3.7 and 4.3.8).

4.3.10 First development on computer. The first time software is developed for a new computer or the first time the developer uses a computer, it can be expected that costs will increase.

- If this is the first time software is developed for a new computer or the first time the developer has used this computer to develop a software program, anticipate a slippage in schedule and increased costs.
- If this is the first time the developer has used the computer to develop a software program, consider having the developer use a computer with which he is familiar, but also consider impact of the factor--different development/target computers.
- If this is the first time a prospective developer would use this computer, suggest consideration of other prospective software developers who have had experience on this computer with similar applications, considering all other things equal.

4.3.11 Concurrent development of hardware. If the software is being developed concurrently with ADP hardware, determine what percent of the software is affected by this concurrent hardware development.

- If less than 10 percent of software is affected by the hardware development (90 percent of software can be developed without effect), the effect will be minimal.
- If greater than 10 percent, expect increased costs.

Determine if off-the-shelf hardware can perform the function of the development hardware.

- If yes, assess resultant software and hardware cost impacts, as appropriate.

- If no, determine if off-the-shelf hardware can be adapted or modified for use. (Assess the cost impact of hardware and software modifications, as appropriate.)

4.3.12 Time CPU specified in schedule (hardware constraint). The later the CPU (or CPUs) is specified in the development schedule, the larger the impact on software development costs.

- In many large weapon systems developments, software turns out to be on the critical path, since major efforts on the software cannot start until the source selection for hardware has been completed.
- Guidelines to cushion the impact of the time at which CPUs are specified in the schedule are as follows:
  - Specify in the performance specifications which CPU is to be used.
  - Force the hardware contractor to select from standard military hardware, thereby greatly reducing software development uncertainty.
  - Develop as much software as possible in standard HOLs, thereby greatly reducing hardware dependence.

4.3.13 Requirement for innovation. This factor includes special displays, concurrent development of other ADP components, a new CPU either development or target, and new languages.

- Anything that falls into the category of being new or innovative will have an adverse impact on software development costs.
- See appropriate discussions of control techniques.
- If existing hardware, techniques, or languages can be substituted for new innovations, then the proposer of the innovations should show cause why the new innovations are required, and tradeoffs should be considered.

4.3.14 Language requirements. The percentage of High Order Language (HOL) versus Machine Oriented Language (MOL) cannot be an edict to software development. If it could, one would simply specify that 100 percent

of the development should be in HOL. For certain types of software, HOL generates intolerably inefficient object code in both the time and memory domains on the target CPU. When this is the case, MOL is the only choice. With this consideration in mind, the following steps should be taken:

- For each function to be performed, assess the efficiency requirements. If the particular task can be performed efficiently by an HOL, select the HOL for the function. If not, assign it for an MOL implementation. For larger more powerful main frames, the necessity for MOL implementations decreases. This is especially true for command and control applications. On the other hand, for minicomputer and micro-processor implementation, the necessity for MOL is increased. This is especially true for on-board flight programs in avionics.
- Once the functions in the development have been categorized into HOL and MOL implementation, a better estimate will be available for the amount of source code required, and thus a much better estimate will be available for the cost to develop.

For command and control applications expect a high degree of HOL implementation. For on-board flight programs in avionics applications expect a low degree of HOL implementation. For simulation and ATE in avionics applications expect a higher degree of HOL implementation, but not as high as for command and control.

4.3.15 Quality requirements. It is recommended that the attributes of quality, defined in paragraph 3.3.1.1.10, be evaluated as part of the development process.

Correctness will be assessed in Verification and Validation (V&V) testing. Ensuring that the software will perform the functions as required will increase the testing costs. This cost will be proportional to the number and degree to which functions will be tested.

Reliability assessment will involve significant testing. The benefit of this testing to ensure reliability objectives have been attained is a decrease in maintenance cost.

Increasing software operating efficiency is a very costly proposition, since it usually requires rewrites to increase efficiency, or going to MOL. It also has a negative effect on most other quality factors, thus increasing life-cycle costs. Since CPU time and memory constraints usually imply the necessity for efficient coding, further guidance can be attained by referring to discussions of the time and memory constraints. Since attaining efficiency usually has an adverse impact on life-cycle costs, only attempt to obtain the absolute minimum level required. System growth may cause the efficiency to decrease, violating minimum acceptable levels. Anticipated growth should be accounted for in the initial design requirements, thereby decreasing the likelihood that recoding will be required during maintenance to attain specified efficiency levels.

Increasing integrity (security) will increase the amount of code required to meet the same set of operational requirements. This will increase development costs, but decrease operational costs. Integrity essentially characterizes how sensitive the system is to operator or system error caused by hardware malfunctions. The project manager has to ask the question, "How much down time due to operator or system error can be tolerated in an operational environment?" If a relatively large amount can be tolerated, then a great amount of integrity is not required in the software design. If only a relatively small amount can be tolerated, then a large amount of integrity should be required in the software design. The amount of integrity required for certain classes of on-board flight programs in avionics is very high. For simulation and ATE for avionics, much less is required. Command and control usually falls in between these two extremes.

Usability refers to how well the software satisfies user requirements. For guidance reference should be made to the discussion of user requirements.

Increasing maintainability will increase development costs, but decrease maintenance costs. Guidance is presented in the discussion of maintenance costs.

Increasing testability will increase the amount of code required to meet the same set of operational requirements. This will increase the cost of the analysis and design, and coding and checkout phases of development, but decrease cost in the testing and integration phase. The amount of testability required should be a function of the size of the development. It should increase with size.

Increasing flexibility (i.e., making the software more adaptable to changing requirements) will increase the amount of code required to meet the same set of operational requirements. This will increase development cost, but decrease maintenance costs. The project manager should analyze flexibility requirements in terms of the expected volatility in operational requirements. If the volatility of operational requirements is expected to be low over the life of the system, then great flexibility is not required. If the volatility of operational requirements is expected to be high over the life of the system, then engineer a large amount of flexibility into the software design.

Portability refers to the ease with which the developed software can be transferred from one hardware configuration and/or software environment to another. Reusability refers to the ease with which the developed software can be used in other applications. These factors are highly interrelated, and are essentially covered in the discussion of transportability. A feature of reusability not covered under transportability is the packaging and scope of the functions developed. This is essentially the modularity put into the design. That is, can a subroutine easily be lifted out and deposited into another development without a lot of awkward interfacing problems. Increasing this kind of modularity will

increase development costs, but may decrease development costs on subsequent developments. This feature is also related to the following attribute, interoperability.

Interoperability refers to the ease with which the developed software can couple with another system. Increasing this attribute will increase development costs, but increase the potential use of the system and also possibly its life. Increasing portability and reusability will increase interoperability; therefore, the discussion on transportability applies. A high use of HOL will increase interoperability. Other features that will increase interoperability are:

- use of standard widely used communications protocols,
- use of standard character representation such as ASCII, and
- use of standard 32-bit and 64-bit formats for floating point representation.

4.3.16 Reliability requirements. Higher reliability means higher development costs, but at the same time can result in lower maintenance costs. There are currently no standard accepted definitions of software reliability. However, meeting reliability requirements, by whatever definition used, affects the cost of the testing and integration phase of development. The higher the reliability, the higher the cost for testing and integration.

Break up the software into reliability categories (classes). Expect higher cost per unit line of code for high reliability categories than for low reliability categories. Assess the reliability required in terms of the failure rate that can be tolerated operationally for each category.

For avionics, reliability requirements should be as follows from high to low: on-board flight programs, simulation, and ATE. Also, life critical software for on-board flight programs should be more reliable

than non-life critical programs. Reliability requirements for command and control should usually fall between on-board flight programs and simulation in avionics.

Several software reliability models exist which may be used for testing the reliability of software development projects. Their applicability and accuracy are discussed by Sukert [119].

4.3.17 Testing requirements (including verification and validation). The imposition of specific testing requirements, such as independent verification and validation (V&V), can increase development costs, however, testing should improve the quality of the software and subsequently reduce maintenance costs.

Independent V&V should increase the quality of delivered software, but expect it to increase total development costs by 20 percent. Independent V&V should probably not be a requirement for small projects, but should be given serious consideration for large projects.

Testing requirements are usually specified in terms of some percentage of logic paths explored. The number of possible logic paths will increase geometrically with the size of developed code. Therefore, expect to explore a greater percentage of logic paths in a small development than a large development. Expect testing costs to be directly proportional to the percent of possible logic paths explored. Although the percent paths explored is currently low, it is expected to increase significantly because of new test tools under development. Do not select the paths at random. Select on the basis of their expected frequency of use in an operational environment. Test those which are expected to occur most frequently.

Since the correction of errors discovered in testing reintroduces the probability of error (i.e., there is a 40 percent chance that correcting an error will reintroduce a new error), regression testing

requirements are sometimes imposed. This involves testing some percentage of the logic paths which are dependent on the path where the initial error was discovered. Follow the same guidance as above for primary paths.

For on-board flight programs in avionics applications, expect to test a high percentage of possible logic paths, especially for software which is classified life critical. For simulation expect the percentage to be lower, and for ATE expect the percentage to be yet even lower. Testing requirements for command and control usually fall between on-board flight programs for avionics and simulations for avionics.

4.3.18 Transportability requirements. Increasing transportability, if the language mix remains constant, will increase development costs. Generally, this cost can only be recouped if there is a change of CPUs over the life cycle or if the code can be transported to other developments. There are secondary cost benefits in training and documentation by using standard versions of standard languages, which inherently makes the code more transportable. Use the following as guidance in assessing transportability requirements:

- Code written in a High Order Language (HOL) is more transportable than code written in a Machine Oriented Language (MOL).
- Code written in a standard version of an HOL is more transportable than code written in a non-standard version.
- Code written in a widely used HOL is more transportable than code written in a less widely used HOL.
- The code required to solve a given problem in a standard version of an HOL will generally be greater than that required in a non-standard version, because the non-standard version is almost always a superset of the standard version, offering the programmer more options in solving the problem.
- Since transportability is almost solely a function of language requirements, see paragraph 4.3.14 on language requirements for additional direction.
- Avionics software is much less transportable than command and control software, since so much of it has to be implemented in

MOL. Most command and control software can be implemented in a standard version of an HOL such as JOVIAL.

4.3.19 Maintenance requirements. Imposition of maintenance requirements will increase development costs, but decrease maintenance costs.

- Maintainability is largely a function of the following factors:
  - language requirements,
  - reliability,
  - testing requirements,
  - transportability, and
  - complexity.

The most important of these is language. HOL is much more maintainable than MOL. Therefore, try to get as much of the development as possible implemented in HOL.

- Another factor affecting maintainability is documentation. Adequate manuals and run sheets for the programs directly affect maintainability. Consider that approximately 30 pages of documentation per 1000 lines of source code will provide average maintainability.

4.3.20 Development schedule. Two curves were developed which demonstrate the relationships indicating expected software development time as a function of object instructions, and projecting expected resource expenditure as a function of planned schedule completion in the analysis of factors affecting software cost estimation. These curves, Figures 5 and 6, were derived from historical data and aid in the management of software development. The curve in Figure 5 assists management in the selection of an adequate time period for software development, and the curve in Figure 6 aids management in the cost and schedule control function.

4.3.21 Communications. In some developments the developer, purchaser, maintainer, and end user are all separate and distinct parties. In others, two or more of the functions may be combined and performed by a single organization. For example, the purchaser and end user may be the same party. For software developed by the Air Force Systems Command,

each function is usually performed by a separate party. The developer is usually a contractor, the purchaser is a SPO residing in the Systems Command, the maintainer is the Logistics Command, and the end user is an operational command such as the Strategic Air Command (SAC).

The developer has to satisfy the requirements of the purchaser, maintainer, and end user. It is usually the responsibility of the purchaser that the requirements of the maintainer and end user get communicated to the developer.

Direct contact between the maintainer or end user and the developer should be discouraged. The Air Force has set up guidelines and procedures to ensure that the end user's and maintainer's requirements are communicated through the purchaser to the developer. These guidelines and procedures should remain high on the SPO awareness scale because of their importance in effecting software development.

4.3.22 Developer using another activity's computer. In the event the development computer is operated by another activity, it can be expected that development costs will be higher than if the developer utilized his own computer.

- If the software developer is using a computer at another facility (e.g., at a government facility) for the software development, anticipate lower programmer productivity (higher costs).
- If software development is being performed by a computer at the developer's facility, anticipate no impact on costs or schedule.

4.3.23 Programmer access to computer. Submitting programs to be tested and run by a separate operations staff is more productive than allowing programmers direct access to the computer.

- This factor only applies to batch environments on large main frames. It does not apply to time-sharing or where the development CPU is either a mini- or microcomputer.

- If the development CPU is a large main frame operating in a batch environment, then
  - a developer who has a separate operations staff and limits programmer hands on CPU availability should be put in a more favorable light than one who does not;
  - in a purchaser on-site cost monitoring situation the amount of programmer hands on CPU checking and testing should be accounted for as much as possible;
  - attempt to keep the programmers confined to programming and preparing test runs, and let the operations staff make the runs on the computer;
  - if a large percentage of machine checkout and testing is done by programmers instead of the operations staff, then expect lower productivity per unit line of code and concomitant higher cost.
- There is a fine balance between bench checking and machine testing programs in terms of achieving optimum productivity. Two runs per day in a batch environment seem to be about optimum for machine testing.
- This factor is of small importance in avionics since one is usually in a minicomputer or microcomputer environment.
- In command and control applications, where batch environments are common, programmer hands-on availability can become a significant problem.

4.3.24 Operational site development. Consideration should be given to the location at which the software development effort is to be accomplished.

- If software is to be developed at the developer's facility rather than at an operational site, anticipate lower costs.
- If the software is to be developed at an operational site (government facility), re-evaluate the requirements for this because of the associated increase in cost (e.g., security requirements, SPO requirement for an on-site interface, joint development, etc.).

4.3.25 Development and target computer different. In the event the target computer is different than the computer on which the software is

developed, it can be expected that costs will increase. If target computer is same as that on which development is to be performed, no effect is anticipated.

- If computers are different, then the following steps should be taken:
  - If adequate support software is not available for the target computer, but is available with the developer's computer, utilize the developer's computer.
  - If adequate support software is available for the target computer, have the developer show cause why the software is not being developed on the target computer.
  - Be prepared to accept increased costs and schedule slippages (including the development of support software for the target computer).
- If coded on a large computer, expect slightly more efficient coding (on smaller computers, expect less efficient coding). This is a potential reason for having the development and target computer different.

Regarding the target computer domain, as one moves toward larger, more powerful main frames, the likelihood of adequate support software increases, thus increasing the likelihood that the target and development computer will be the same. This is especially true for command and control applications. As one moves toward minicomputers and microprocessors, the likelihood of adequate support software decreases, thus increasing the possibility that the target and development computer will be different. This is the case for avionics applications.

4.3.26 Number of development locations. When software is being developed at more than one location, it can be expected that development costs will increase and the following should be considered:

- Have the developer justify multi-site development. Peculiar end user requirements where each installation has site dependent software and the developer has to be on-site may be an acceptable justification. Causes associated with the internal corporate structure of the developer will be harder to justify.

- A developer who proposes a single site development should be put in a slightly more favorable position than a developer with the proposed software team spread over many locations.
- Examine the feasibility of having the developer bring his entire proposed software team together at one site.
- For large systems the likelihood of multi-site development increases. In many cases, avoiding it will be impossible.
- If multi-site development occurs, expect increased costs and a minor slippage in schedule.

4.3.27 Programming environment. The programming environment created by modern programming techniques, such as the imposition of certain structured programming disciplines, can decrease cost over more traditional methods. Other elements not associated with structured programming contribute to programming environment. Their impact is expected to be large, but the magnitude is not known. Certain structured programming techniques that fall within the realm of helping to create programming environment, such as Chief Programmer Teams and Programming Support Libraries, can have a beneficial effect on software costs. These techniques have their maximum impact on medium sized projects, approximately 100,000 source lines. Their impact lessens for very small or very large projects.

The Chief Programmer Team concept can be implemented without incurring any additional development costs, and probably should be done for all medium sized projects. If Programming Support Libraries are available on the proposed development computer, then they probably should be used, at least for medium sized projects.

If they are not available, then one should assess the feasibility of developing them out of project funds. For small projects, it is probably not worth it. For medium and large projects, one should trade the cost of developing them off against the percent savings in development expected (maximum of 40 percent).

92

To realize the full benefits of these techniques, compilers and pre-compilers that accommodate structured code must be available. If they are not available, then the tradeoff between benefits and paying for their development should be examined.

Other elements that contribute to programming environment are test tools such as support software for debugging, loaders, and other kinds of utilities. Software productivity will be directly related to the availability of these tools. If they are not available to an adequate degree, then the program manager should examine the tradeoff of paying for their development versus the lower productivity expected without their availability.

4.3.28 Support software. The availability and quality of support software can have a considerable impact on development costs.

- If either the development computer or the target computer or both are new, expect to pay a considerable amount for support software compared to a development where these conditions do not exist. If possible, try to avoid using a new computer as either the development or target machine. If a new computer is chosen, the advantages it provides should clearly outweigh the additional cost that will be required to develop adequate support software.
- If the target computer is a large main frame expect the quality and availability of support software to be good.
- If the target computer is a minicomputer or microprocessor expect the quality and availability of support software to be poorer.

4.3.29 Programming facilities. The availability and quality of programming facilities, such as office space and layout, and computer accessibility and support personnel, have a considerable impact on development costs.

- Let the developer control the programming facilities to as high a degree as possible. This includes:
  - development at the developer's site instead of a purchaser selected site, and
  - development on a developer controlled dedicated computer instead of a computer run by another organization.
- There may be extenuating circumstances where this is impractical. For example, the cost of supplying the developer with the development CPU may be large compared to making time available to the developer on a non-developer controlled computer.

4.3.30 Multiple software utilization sites. Developing software for multi-site utilization is less productive than developing software for a single site utilization.

- If the multi-site software to be developed has no site dependent features, then expect no impact from this factor.
- If the multi-site software to be developed has site dependent features, then expect the cost to increase by the number and size of such features to be implemented.
- If the multi-site software to be developed has inter-machine communications, expect the cost per unit line of code to be higher than if no inter-machine communication is required.

4.3.31 End user requirements. The degree to which the user requirements are defined accurately in the performance and design specifications is a determinant of the software acceptability.

- The project officer or SPO should be completely aware that the development will be deemed unacceptable if it does not meet user requirements. Therefore, the end user should be involved as much as possible in the development.
- The Air Force has guidelines and procedures to ensure that end user requirements are available to the developer, via the SPO [AFR 800.14 (AFSC SUP)].

4.3.32 Design complexity. The complexity of a software project will greatly affect the programmer productivity. General rules of thumb to keep in mind in this venue are:

- Operating systems are more complex than compilers, and compilers are more complex than applications software. Support software in general is more complex than applications software.
- Real-time applications are more complex than non-real-time applications.
- Interrupt driven multi-tasking software is more complex than non-interrupt driven single tasking software.
- Scientific applications are usually more complex than business applications.
- On-board flight programs for avionics are often more complex than command and control programs.
- Simulation and ATE for avionics are about equal in complexity to command and control.

Complexity can be looked upon as an overview of a number of items covered by other factors. Once the design has been approved, after CDR, the project director should consider breaking up the software by levels of complexity, and costing each portion separately.

4.3.33 Design stability. Instability of design can result in increased developmental cost.

- Since 60 percent of the errors discovered in testing may be caused by faulty design, it should be expected that there will be some instability in the design. There is no ironclad way of ensuring an initial stable design.
- The use of formal requirements analysis and design languages, if available, as discussed under the factor, Definition of Operational Requirements, may tend to increase design stability.
- The use of modern programming techniques may also increase design stability.
- For large projects, design changes are probably inevitable; therefore, allow some flexibility in both schedule and cost to account for this eventuality.

4.3.34 Modern programming techniques. The maximum benefit from modern programming techniques will be realized from mid-sized projects, approximately 100,000 lines of source code. Its effect is lessened for

smaller and larger sized projects. The benefit derived from the use of modern programming is a function of the number of associated disciplines imposed, e.g., Chief Programmer Teams, Programming Support Libraries, and Hierarchical Input-Output (HIPO).

The imposition of some of these disciplines may involve additional investment costs. For example, the development computer proposed may not provide Programming Support Libraries. In this case, an investment would have to be made in software development to provide Programming Support Libraries for the development computer.

See paragraph 4.3.27 on Programming Environment for an examination of these tradeoffs.

4.3.35 Sizing estimate error. The use of the cost per instruction method for the cost estimating will cause sizing error to have a direct impact on the error in the cost estimate.

- The sizing error will get smaller as the project moves toward completion.
- Since the error associated with productivity (instructions per man-month) changes as a function of the instruction count parameter, the sizing selected is dependent upon the development phase in which the cost estimate is being made. Use the following guidelines for selection:
  - Conceptual Phase - Initial Budgetary Estimate,
    - Total size in object words (up to 200 percent error)
  - Validation prior to release of RFP,
    - Size in object words minus data areas (up to 100 percent error)
  - After receipt of proposals through PDR,
    - Size in new object words minus data areas after adjustment for reusable code (up to 75 percent error)
  - From PDR through remainder of development,
    - Size in new source statements (up to 50 percent error improving to completion)

4.3.36 Definition of instruction. In order to obtain a reliable estimate from the cost per instruction method, the definition on which the instruction count is based must be consistent with that used in developing the Cost Estimation Relationship (CER). Guidelines to ensure consistency between instruction count and the CER for cost estimation are as follows:

- If the CER was developed on the basis of object code, then
  - instruction count should be in object code,
  - handle data areas and constants consistent with the CER,
  - handle reusable code consistent with the CER,
  - handle deliverable versus non-deliverable code consistent with the CER, and
  - handle support versus operational software consistent with the CER.
- If the CER was developed on the basis of source code, then
  - instruction count should be in source statements;
  - handle comment, copy, and declarative statements consistent with the CER;
  - handle reusable code consistent with the CER;
  - handle deliverable versus non-deliverable code consistent with the CER; and
  - handle support versus operational software consistent with the CER.

4.3.37 Personnel mix by type. The expected mix of support personnel (management, clerical, etc.) to programmers and analysis is 20 percent support personnel to 80 percent programmers/analysts. Deviations from this mix depend on software project peculiarities. If the developer has a mix sharply different from the expected, have the developer show justification for the mix, or consider that for each 10 percent increase in support personnel relative to programmers/analysts, productivity will decrease by 25 percent.

4.3.38 Work Breakdown Structure (WBS). The structure of the Work Breakdown Structure (WBS) is critical in measuring the actual development cost of software for an embedded computer system. The WBS for a system with embedded computers will contain much more than elements related to software. Systems with embedded computers are the general rule for command and control and avionics applications. Therefore, it is essential that the WBS be constructed properly to get the actual software cost. Guidelines to ensure that software gets reflected properly in the WBS are as follows:

- A single element in the WBS for software will very seldom account for the total software development cost. Usually a single element in the WBS for software will account for coding and check-costs, nominally 20 percent of total software effort.
- It is imperative that analysis and design, and testing and integration get reflected in the WBS. Since the WBS for most systems with embedded computers will be oriented around prime mission equipment elements, the portions of each prime mission equipment element targeted for software implementation must have separate software elements for analysis and design, and testing and integration.
- Make sure that management and support costs for software development are adequately reflected in the WBS. In order to do this, it is usually necessary to include software elements in the System Engineering/Project Management portion of the WBS. These elements should be partitioned by the software life-cycle phases.
- Factoring hardware from software in a satisfactory manner in the WBS is very difficult for many developments. Many engineers, especially in avionics applications, are dually qualified in both hardware and software. Partitioning their time accurately among the various WBS elements is very difficult. This is especially difficult in the testing and integration phase, since the cause of many problems encountered cannot be attributed to hardware or software until they have been resolved. The only solution appears to be constant watchdogging to ensure that labor gets partitioned accurately among hardware and software elements in the WBS.

4.3.39 Cost/Schedule Control Systems Criteria (C/SCSC). The C/SCSC, or its non-formal equivalent, is the principal mechanism for determining

if the project is deviating from planned cost and schedule. The reporting vehicle for C/SCSC is the Cost Performance Report (CPR). C/SCSC will only be as effective as the Work Breakdown Structure (WBS) for the development. If only one software element is in the WBS, then C/SCSC is not going to be a very effective mechanism for controlling software costs. See paragraph 4.3.38.

Constant surveillance of cost and schedule variance should be maintained. There are a number of techniques for analyzing cost and schedule variance to affect project control which should be implemented.

A simplified rule of thumb, which can be used as another control mechanism, is the 40-20-40 rule: 40 percent of development effort in analysis and design, 20 percent in coding and checkout, and 40 percent in testing and integration. The first of these development phases has a milestone usually associated with it, the Critical Design Review (CDR). If the CDR, which reflects the end of design, has not been completed by the time 40 percent of the funds have been expended, there could be a potential cost overrun.

4.3.40 Amount and method of cost data collection. This factor has no impact when the existing cost data collection system in place at the developer's facility is considered adequate. The amount of data to be collected is a direct function of the Work Breakdown Structure (WBS) and the Cost/Schedule Control Systems Criteria (C/SCSC) set up for the development. When data is required that is not part of a developer's ordinary cost data collection practices, then the following should be considered:

- It should be determined if the development costs to implement new cost data collection practices can be recouped during the current development. If not, benefits to subsequent software developments should be considered.

- New cost collection practices could be implemented by manual or automatic means. Manual implementation will cost less to develop than automatic, but will carry a higher operational cost. The tradeoff should be examined carefully, especially relative to benefits to subsequent developments. Some cost collection practices will not be easily amenable to implementation by automatic means.

4.3.41 Secondary resources. Secondary resources, primarily computer time and documentation production, on the average, will amount to 15 percent of the total software development costs.

- On the average, about 4 to 5 hours of computer time per man-month should occur in a batch single partition environment. While costs will be equivalent, time will not be the same in terminal oriented or multi-programming environments.
- On the average, about 5 pages of documentation should be estimated per man-month.
- If secondary resources are to be accounted for accurately, then the WBS has to be structured properly to account for them. Any algorithm used to estimate secondary resources should be consistent with the WBS. For example, if the algorithm estimates only computer time and documentation production costs, then the WBS should have specific elements for these items. Often, secondary resources will be buried in overhead or other non-software elements in the WBS. In these cases it will be virtually impossible to measure the actual cost of secondary resources. Therefore, any algorithm used to estimate these costs will prove of little use.

#### 4.3.42 Software applications.

4.3.42.1 Business. Most business applications for the Air Force are implemented under the control of the Air Force Data Systems Design Center (AFDSDC). The primary language used is COBOL. A formalized procedure for development exists, and is documented in Design Center manuals. It covers all life-cycle phases from analysis through operation. A management information system for resource planning and utilization exists called PARMIS (Planning and Resource Management Information System).

Most business applications can be implemented in either COBOL or RPG. It is difficult to justify implementation in an MOL, since time or memory efficiency requirements, the major reason for MOL implementation, are seldom severe. Since these applications generally have a high degree of I/O relative to computation, sizing or costing algorithms based on the number of I/O items can be quite effective. A number of business application programs are written on the basis of transaction oriented processing to update files. In these cases the number of transactions can serve as an estimator of size and cost.

4.3.42.2 Scientific. Most scientific applications, except in a real-time environment, can be implemented in an HOL. The widely used languages oriented around batch development are FORTRAN, ALGOL, PL/I, and JOVIAL. PL/I has the additional advantage of having features which are applicable to business applications. The widely used languages oriented around interactive development are BASIC and APL. It is difficult to justify the use of an MOL for scientific applications in anything other than a real-time environment. For real-time scientific applications, refer to the discussion on response time, and CPU time and memory constraints (pars. 4.3.6 through 4.3.9).

Probably the largest scientific developments in a non real-time environment are Monte Carlo simulations. If the simulation is event driven, then the number of events can be used to estimate size.

4.3.42.3 Command and Control. The applications are often large, about 500,000 object words on the average. This large size causes low productivity per unit line of code. Most of this software is targeted for large main frames; therefore, the potential for excellent support software exists, especially for main frames that have been in the field for a long time. Most applications can be implemented with a high degree of HOL usage. The standard Air Force language is JOVIAL. Developers who propose a small degree of HOL usage should show cause why.

4.3.42.4 Utility. Developing support software that operates in an on-line or utility mode is more productive in terms of cost per unit line of code than development software in general.

4.3.42.5 Avionics. Avionics software is divided into three categories: On-board flight programs, Simulation, and ATE. On-board flight programs will be least productive because of time and memory constraints, and extensive testing. Simulation will be second least productive, and ATE most productive.

Although there is currently a low degree of HOL implementation for on-board flight programs as compared to simulation and ATE, the trend is toward greater utilization of HOL. A convenient size estimator exists for ATE; expect about 1500 lines of source code for each Line Replaceable Unit (LRU).

## 5. METHODOLOGY FOR ESTIMATING SOFTWARE DEVELOPMENT COSTS

The objective of Task 3 of the effort was to integrate the results of Tasks 1 and 2 into a methodology for improved estimation of software development costs. The methodology incorporates the following into a formalized procedure for deriving more accurate cost estimates: algorithms for estimating software development costs, the effects of various factors on these costs, and controls for the adverse effects. The costs have been categorized as reflecting the primary and secondary resources of development. The primary resources include the time and manpower required to develop the software; the secondary resources include such elements as computer time, documentation, and personnel travel.

The basic approach used to estimate the costs involved the development of baseline algorithms to estimate the labor required for software development, and the structuring of modifiers which adjust the baseline estimates to reflect the impacts of program parameters, the generation of an estimator for secondary resources, and the derivation of an estimator for development time required. Program size was found to be the best determinant of development costs, and since the estimating relationships include program size (lines of source code and/or words of object code) as an independent variable, guides to assist in estimating the size of software have also been incorporated into the cost estimating methodology.

The quality of estimating relationships developed in the statistical analysis of historical data is dependent upon the quantity, accuracy, scope, refinement, variability, and ranges of values reflected in the data base. The data analyzed for this study is considered the best software development data base structured to date; yet, the inadequacies evident in this data base have affected the study results appreciably. The specific effects are noted in the subsequent discussions of each estimator.

## 5.1 Development of the estimating algorithms

5.1.1 Time required for development. The SDC data included information relative to the months elapsed between program delivery and initiation of design. From this data, linear regression techniques were used to derive a relationship for the time to develop software as a function of program size, in object code. Object code was used as the size parameter because program size will probably be known in terms of object code in the initial stages of software development, when program schedules are being planned.

In the initial regressions, the duration of development was found to be proportional to the cube root of the object code. However, the shape of the curve was found not to fit the data for small programs satisfactorily. Consequently, to develop a relationship offering better fit to the data, regressions were made with the cube root of the object instructions in various forms. The relationship offering the best fit was of the following form:

$$D = \frac{I}{a + b I^{0.67}} \quad (\text{Equation 3})$$

where

D is the duration of development, in months

I is the size of the program, in words of object code

a is a constant (= 99.25)

b is a constant (= 2.33)

The coefficient of determination ( $R^2$ ) of this relationship is 0.52. Figure 5 (pg. 44) is a plot of this relationship. Since it reflects historical data, projections made from the curve implicitly assume that the environment reflected in the data will be reflected in future programs.

The duration of software development programs is often dictated by exogenous considerations, such as the time schedule for ADP equipment development, the time at which interfacing weapon systems are to become operational, etc. Consequently, it is unlikely that adequate time is being allowed for large software development programs. Therefore, the data was examined to determine if it would be possible to develop a relationship for optimal (least cost) duration of the development programs. Based on the examination, DAI has tentatively concluded that current data can support the development of such an estimator.

Further examinations of the data indicated that the form of the optimum duration estimator is a function of the number of source lines of instructions and manpower loading. It also appears that manpower and time are not freely interchangeable, and that the software application (command and control, scientific, business, or utility) may influence the optimum duration and associated manpower loading. Examination of derivatives of a duration curve, expressed in terms of program size and manloading, revealed there is a possibility that for large programs the optimal curve may indicate a requirement for more development time than that reflected by the derived relationship, as shown in Figure 8.

5.1.2 Cost estimating algorithms. Most analyses of software costs have resulted in the statistical derivation of estimators of software development costs as a function of software size, i.e., the number of source lines and/or object words. Usually, these relationships are of the following form:

$$C = aI^b$$

where

C is the cost of development,

I is the size of the program in number of source instructions or words of object code, and a and b are constants.

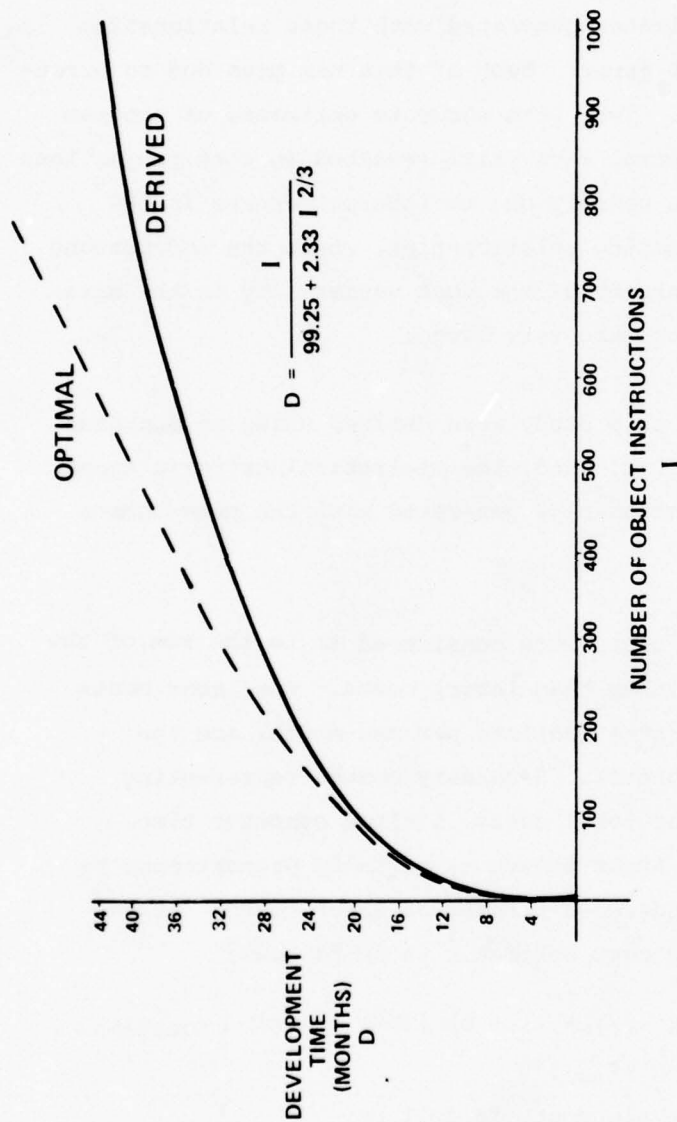


Figure 8. Anticipated shape of estimator for optimal duration of software development

(In almost all cases, the exponent,  $b$ , has been found to be greater than 1.0, indicating dis-economies of scale. However, General Research Corporation [59] has hypothesized that  $b$  could be less than 1, and that the value may approach 0.5.)

Unfortunately, the cost estimates generated with these relationships have been subject to significant error. Much of this has been due to erroneous estimation of program size. Yet, with accurate estimates of program size, the cost estimators in general have still resulted in cost projections with significant error. This is usually due to inherent errors in the statistically derived cost estimating relationships, where the independent variables define less than 30 percent of the cost variability in the data and the associated standard errors are very large.

The algorithms developed in this study were derived using an approach not yet reflected in the literature. And, the statistical criteria appear significantly improved over relationships generated with the more common statistical approaches.

For this study, development costs were considered to be the sum of the primary (labor) and secondary (other than labor) costs. The labor costs are a product of average labor rates (dollars per man-month) and the amount of labor required (man-months). Secondary costs, representing a comparatively small part of the total costs, include computer time, documentation, and other costs; these elements, normally proportioned to manpower requirements, are considered a percent of labor costs. Therefore, the basic structure of the cost estimator is as follows:

$$C = [(MM) \times (r)] + k [(MM) \times (r)] = (1 + k) [(MM) \times (r)] \quad (\text{Equation 4})$$

where

$C$  is the cost of software development, in dollars,

$MM$  is the manpower required for the development, in man-months,

$r$  is the average labor rate of the development manpower, in dollars per man-month, and

$k$  is a factor expressing the fraction of total labor costs considered to approximate secondary costs.

The costs were defined in terms of man-months of development labor because it eliminated the conversion to constant dollars, and users of the model can apply the most appropriate labor rate for their specific problem. For this purpose, the labor rate includes direct labor costs, and associated overhead costs, general and administration (G&A) costs, and fees.

As noted previously, the size of the program, the pages of documentation, and the number of work packages reflected in the software development are the variables most highly correlated in the development manpower. None of these variables are known accurately early in the development process. However, since the size variable is usually estimated more easily (and perhaps more accurately), it usually appears as an independent variable in relationships for development manpower (or cost).

The estimating relationships for development manpower were derived for the total population and various subsets of the populations. For example, estimators were generated for software applications and by size within the applications. This resulted in the generation of models considered more responsive to the user needs, and more accurate for developments within the specific applications.

As noted in Section 3, extraneous data sets were deleted from the data population. Forty sets of data were deleted from the 169 sets in the total population. For the most part, these represented programs in which the delivered code was 40 percent or less of the total written. In a few cases, the data sets included programs reflecting productivity too high to be considered accurate (2000 source lines per man-month or greater). These latter programs may have included recovered code taken from other programs, or the data may have been recorded improperly. Since the data reflects software developed prior to 1966, it is not considered to reflect modern

programming practices such as structured programming, chief programmer teams, etc. All the programs deleted for this reason were business programs; thus, it is likely that the high productivity resulted from the incorporation of previously written code. Table 9 outlines the structure of the data sample in terms of the size and type program. Most of the programs have fewer than 10,000 instructions. The largest program contained 55,000 lines of source code and 55,000 words of object code. (A data set encompassing 225,000 object words and 115,000 source lines was among those deleted.)

TABLE 9. STRUCTURE OF THE DATA SAMPLE

Type Program	Total in Population	Total After Data Deleted	Source Code		Object Code	
			I $\geq$ 10K	I<10K	I $\geq$ 10K	I<10K
Scientific	27	21	6	15	10	11
Command & Control	35	24	7	17	7	17
Business	79	58	4	54	10	48
Utility	28	26	16	10	19	7
Total	169	129	33	96	46	83

Baseline estimating relationships for manpower as a function of program size were derived with the customarily used linear regression techniques. However, the fit of these equations to the data for larger programs was not considered satisfactory. Then, non-linear regression techniques were applied to the data to determine if the fit could be improved. The algorithm used was Marquardt's method of gradient expansion for least-squares fit to an arbitrary function.<sup>1</sup> In all cases, the fit of the baseline relationship was improved significantly. Multivariate linear regression was performed on the variables in the data assessed in correlation analysis to have the greatest impact on the manpower requirements. The variable included for program size was the baseline relationship of

1. Marquardt, Donald W.: An Algorithm for Least-Squares Estimation of Non-linear Parameters, Journal of the Society of Industrial and Applied Mathematics, Vol. II, No. 2, pg. 431-441, June 1963.

the source (or object) code. This permitted an assessment of the impact of the factors on the baseline relationship. (When program size was used in lieu of the baseline relationship, the effects due to the variables did not change appreciably; the significant change was in the coefficient of the equation.) For the relationships derived, MM is the total manpower required for development, in man-months, and I is the size of the program, in words of object code or source lines of instruction, as appropriate. Since the programs were somewhat limited in size, the estimating relationships are subject to the significant error that can occur in extrapolation beyond data in the sample.

In general, the non-linear regression generated preferred baseline relationships; the fits to the larger program data were much better than that obtained in linear regression. Comparisons of actual manpower expended with estimates generated by the multivariate relationships indicated that the estimators are accurate for small programs, but tend to over-estimate requirements for large programs. This conservatism may be desirable for budgeting purposes, but not for estimating resources required. Therefore, the baseline estimators should probably be used for the larger programs where more accurate estimates of the resources are needed.

5.1.2.1 Total population. Figures 1 and 2 (pp. 19, 20) reflect the high dispersion of the data in the total population, and the data sets deleted for reasons discussed previously. Figure 9 presents the baseline manpower relationships for the total population created using linear and non-linear regression techniques with the program size expressed in words of object code and lines of source code. The expressions reflect almost linear relationship with program size.

The fits of the expressions to the data are exhibited in Figures 10 and 11 for source code and Figures 12 and 13 for object code. Figures 11 and 13 show the fits to the smaller programs ( $I < 10,000$ ). In general, the fit to the total population is better with the non-linear regression

	LOG-LINEAR	NON-LINEAR
OBJECT CODE	MM = 3.0951 0.899 (R <sup>2</sup> = .24, SE = * )	<div>MM = 4.7901 0.991 (R<sup>2</sup> = * , SE = 62.2)</div>
SOURCE CODE	MM = 4.3551 0.990 (R <sup>2</sup> = .53, SE = * )	<div>MM = 5.2581 1.047 (R<sup>2</sup> = * , SE = 50.7)</div>

Denotes preferred relationships

\* DATA NOT PROVIDED BY AUTOMATED PROGRAM

Figure 9. Baseline relationships for software programs

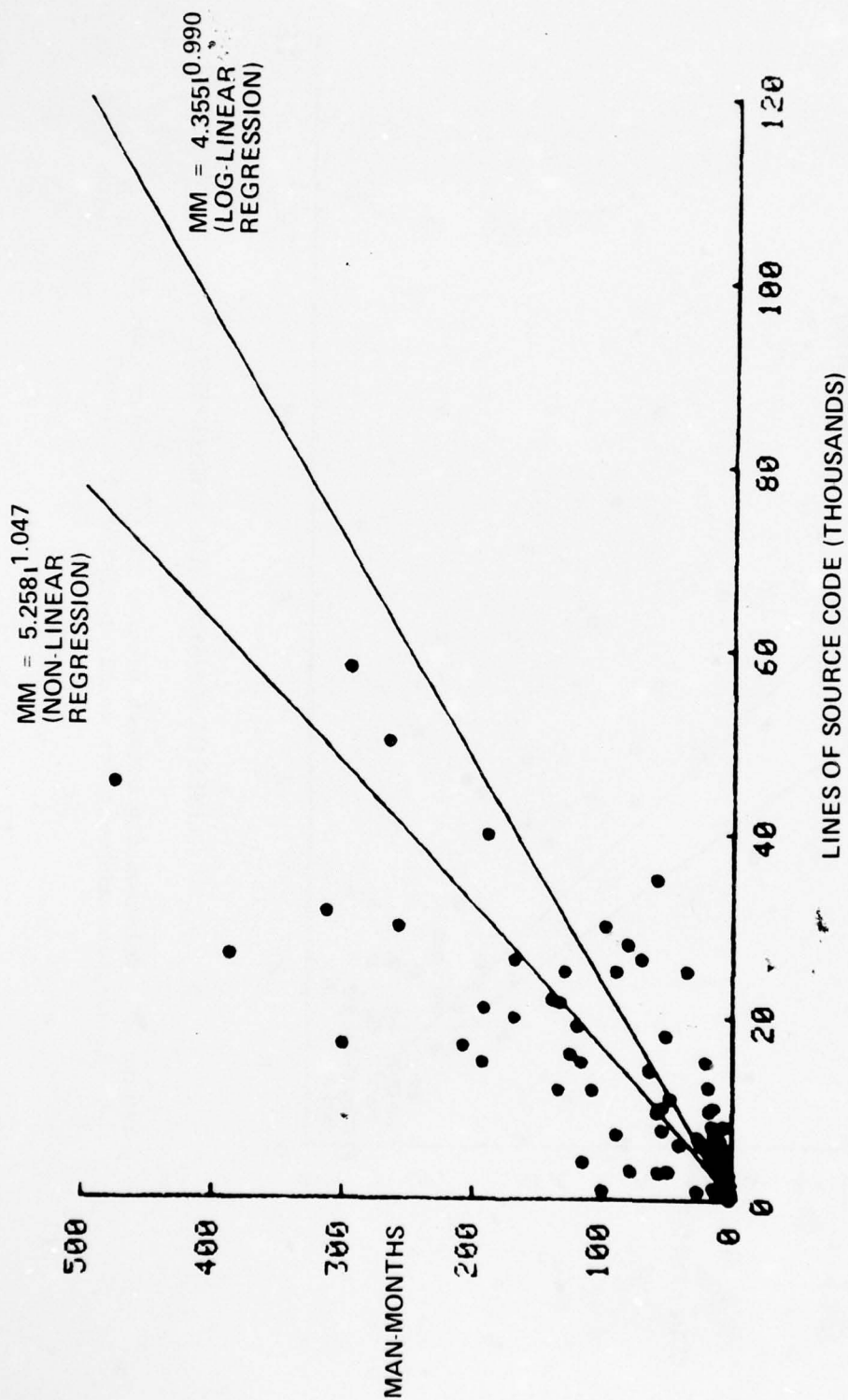


Figure 10. Relationship between program size in source code and development manpower for total population

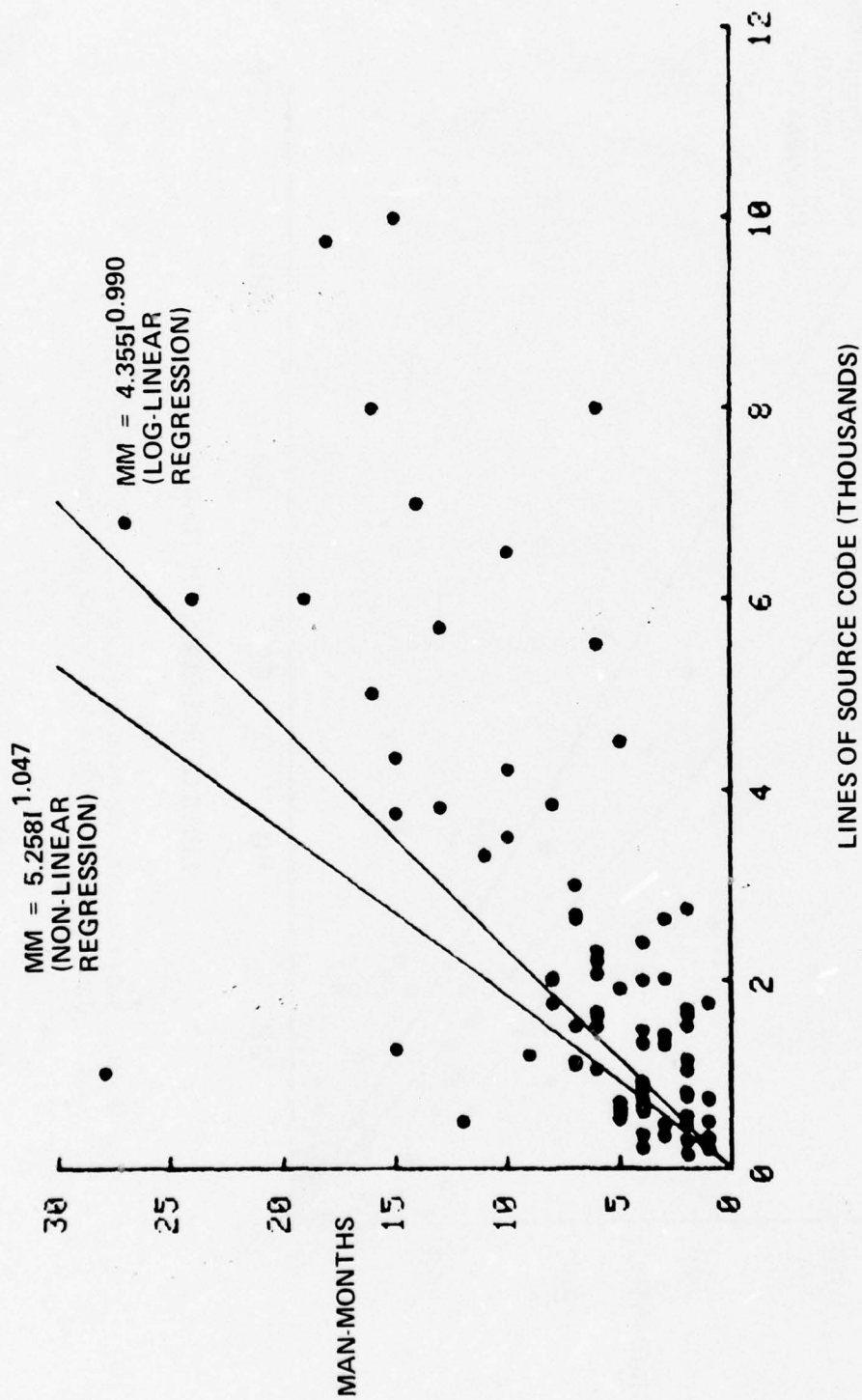


Figure 11. Relationship between program size in source code and development manpower for total population ( $I < 10,000$ )

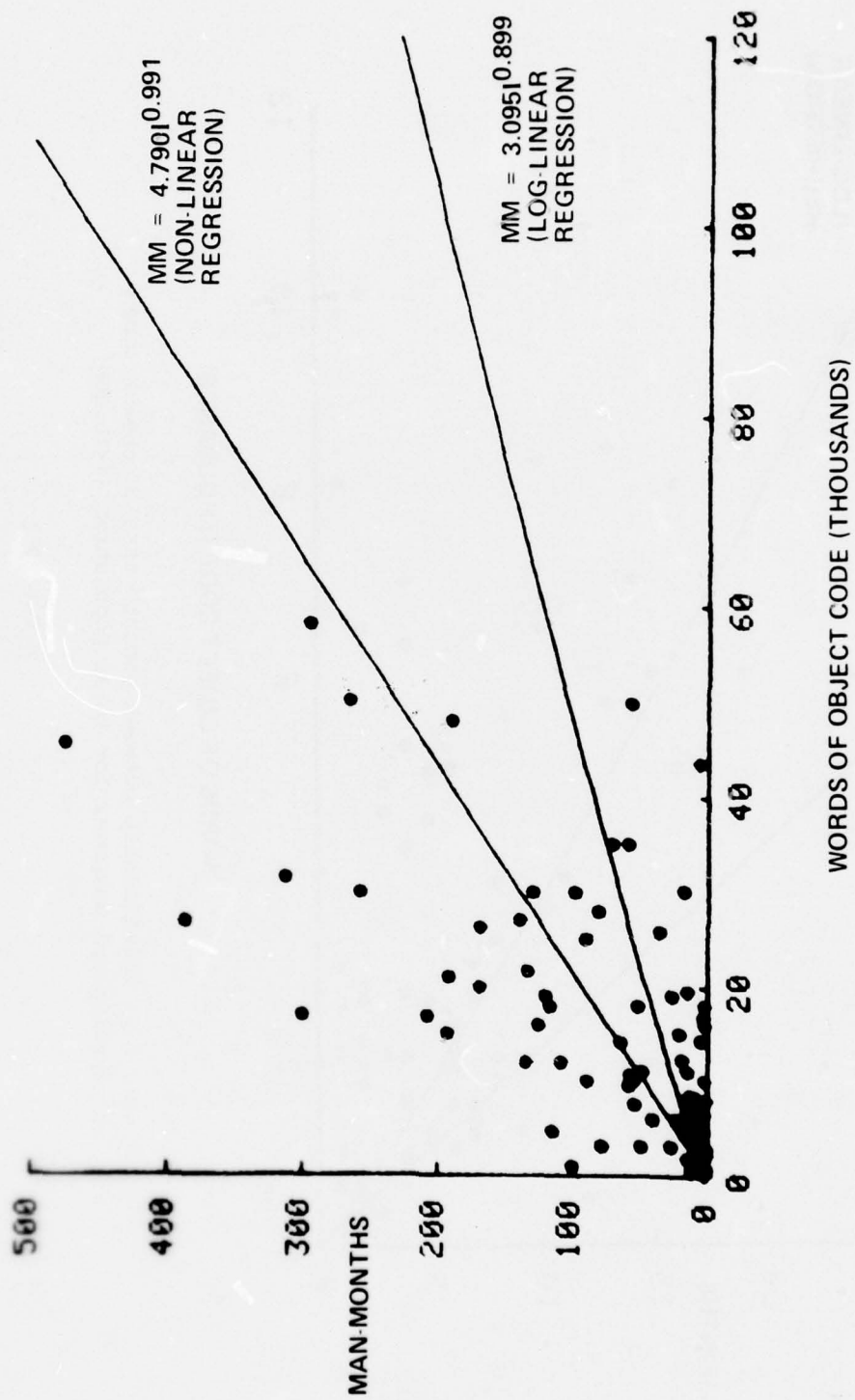


Figure 12. Relationship between program size in object code and development manpower for total population

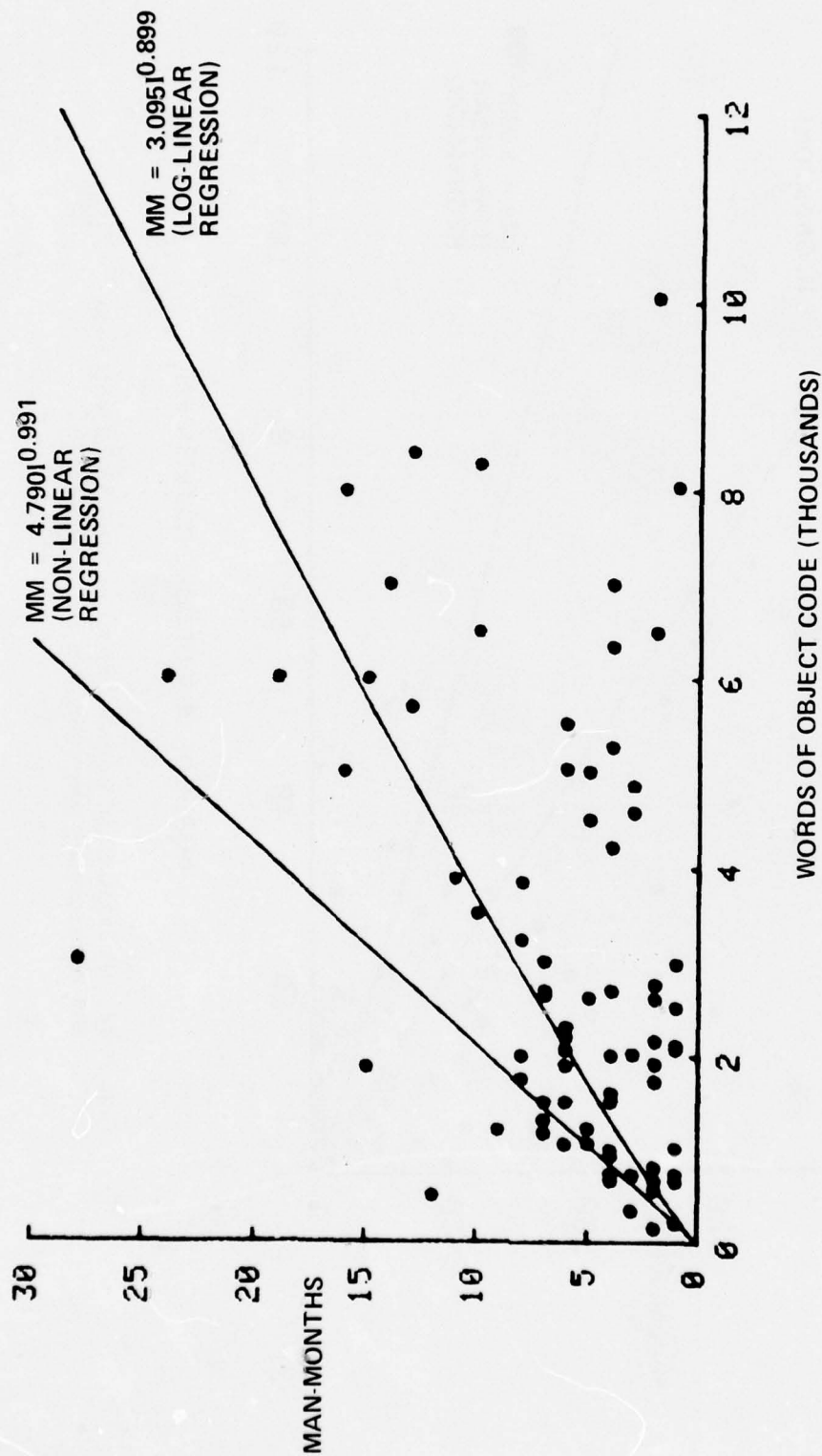


Figure 13. Relationship between program size in object code and development manpower for total population (I<10,000)

models; however, for the smaller programs, the linear regression models appear to offer a better fit. In all cases, the estimates of the non-linear regression models exceed that of the linear regression models. A comparison of actual and estimated development manpower for the source code model is presented in Figure 14.

Figure 15 presents the manpower estimation model for the total population and the values for the state variables  $f_j$  based on multivariate regression of the data. To use this relationship, answers to the yes-no considerations must be projected. The variables,  $f_j$ , essentially represent the magnitude of the effects anticipated relative to the average effects reflected in the data. The coefficient of determination for this relationship exceeded 0.87; the standard error of estimate is 0.800 natural log units.

Because of the better fit associated with the non-linear regression relationships, they are preferred as baseline relationships. However, as more information is determined about the development, the multivariate relationship can be used. In a worse case situation (all variables  $f_j$  at maximum), the manpower estimate from the multivariate relationship will be 12.60 times that of the preferred baseline relationship.

5.1.2.2 Command and control. As shown in Figures 16 and 17, eleven of the 35 command and control programs in the data base were deleted because their percent code delivered was less than 40 percent of the total code written. Most of the data sets are the same in source and object code indicating that the code was probably written in machine language. The largest program in the command and control sample is 30,000 lines of source (or object) code. (Only seven of the twenty-four remaining data sets involved programs of 10,000 or greater lines of source code.)

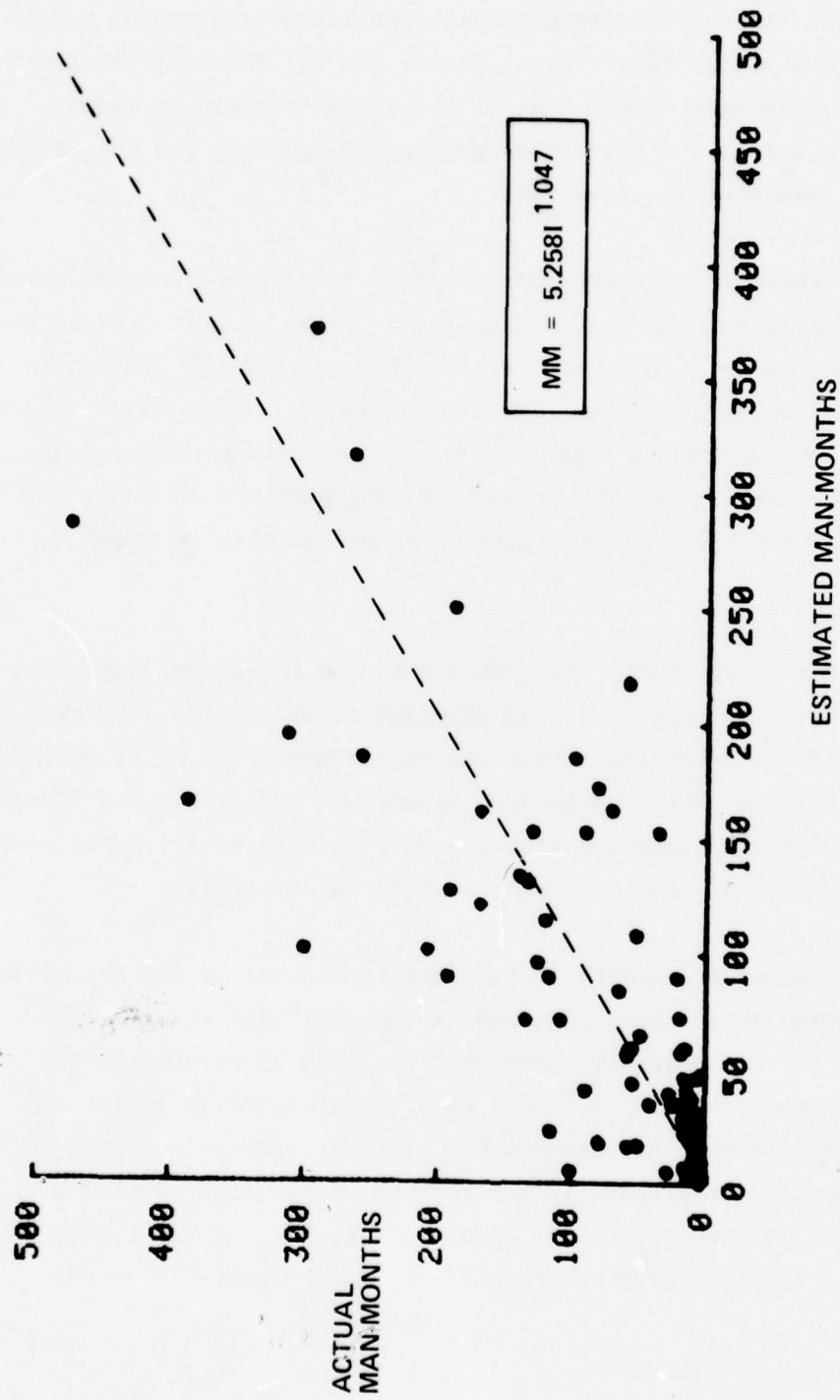


Figure 14. Comparison of actual and estimated development manpower for total population



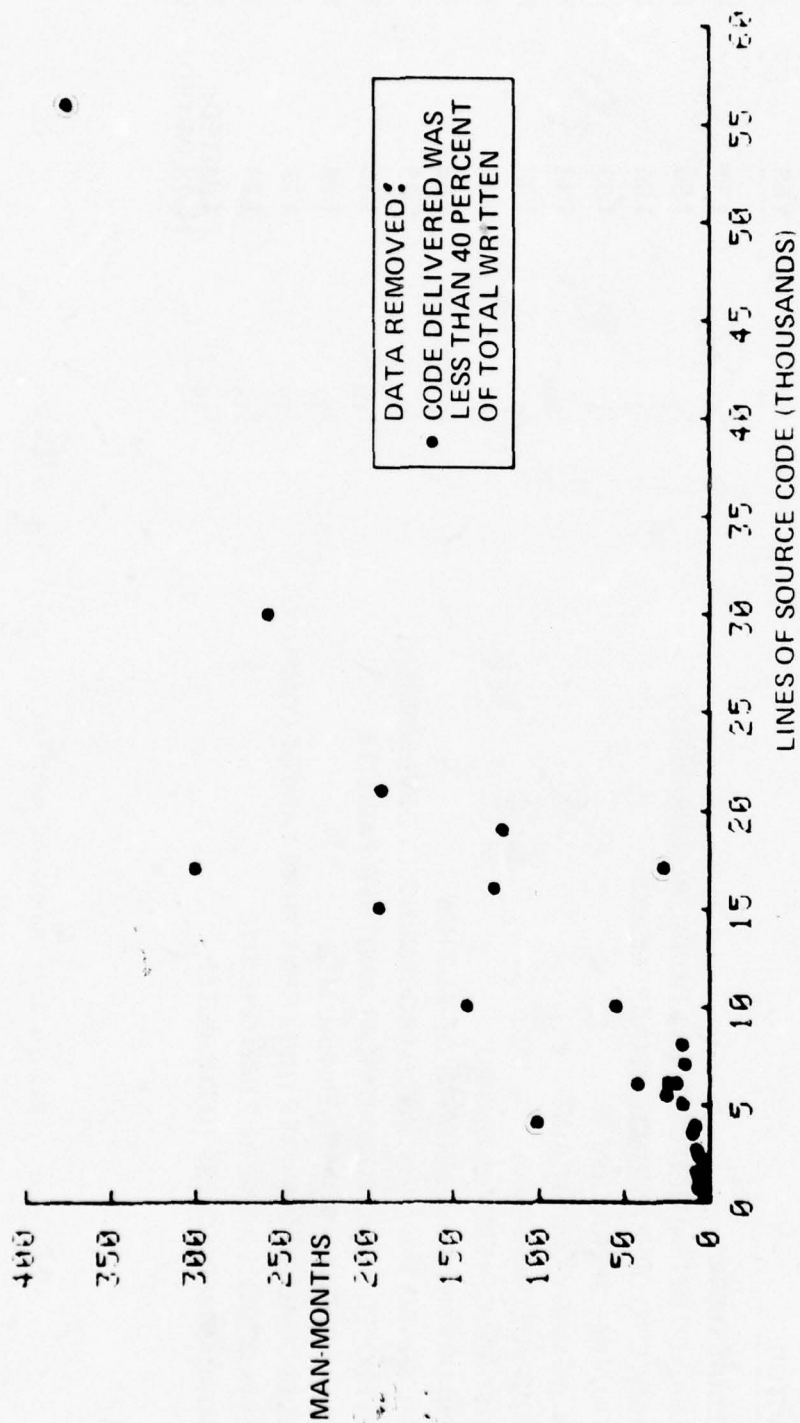


Figure 16. Command and control program source code data

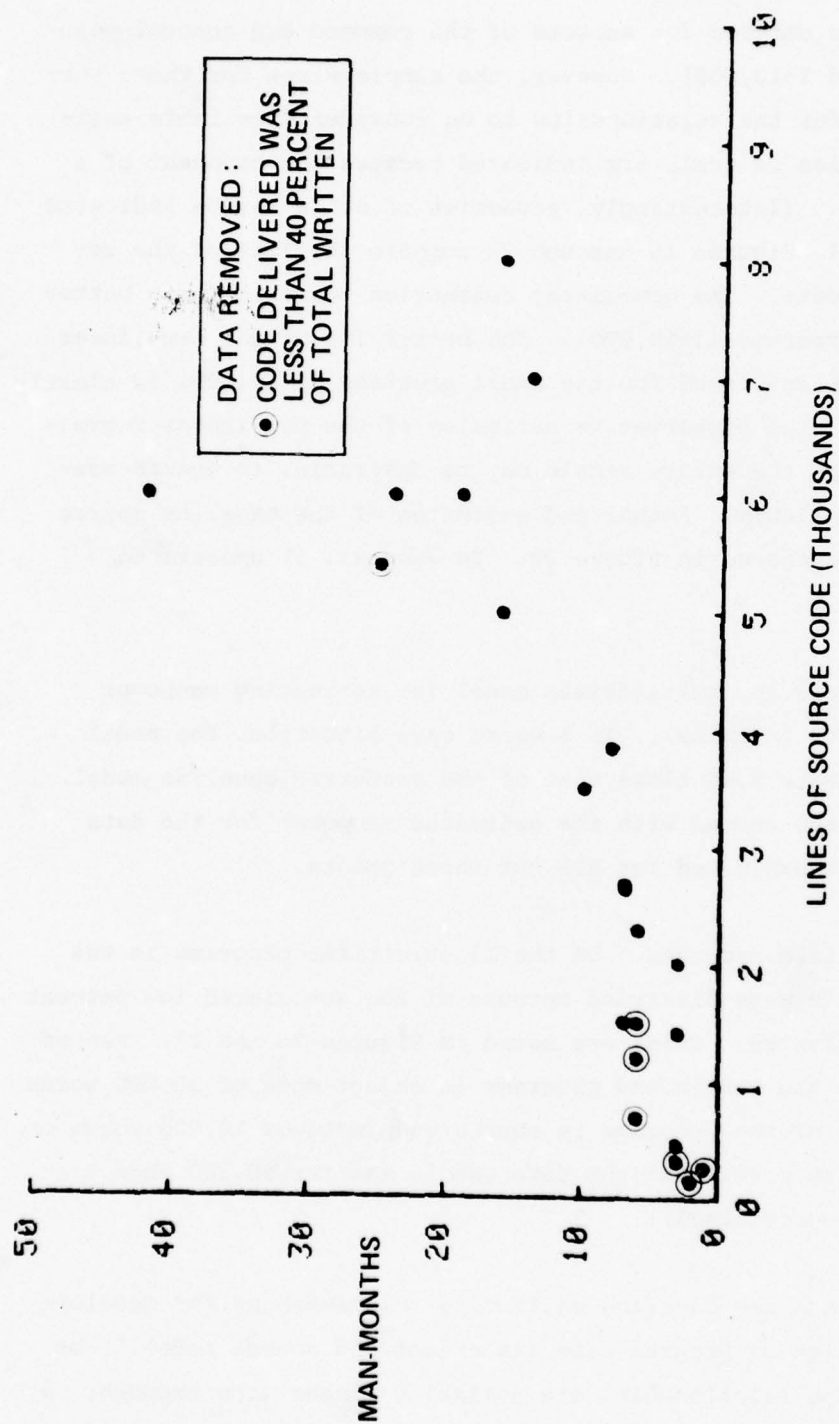


Figure 17. Command and control program source code data ( $I < 10,000$ )

Figure 18 presents the baseline relationships for this application. Estimators were also derived for subsets of the command and control population ( $I \geq 10,000$  and  $I < 10,000$ ). However, the sample sizes for these subsets are too small for the relationships to be considered reliable estimators. Dis-economies of scale are indicated because the exponent of  $I$  is greater than 1.0. (Interestingly, economies of scale may be indicated for small programs.) Figures 19 through 22 compare the fits of the relationships to the data. The non-linear regression curves offer a better fit to the larger programs ( $I \geq 10,000$ ). The better fit of the non-linear regression estimator developed for the small programs ( $I < 10,000$ ) is clearly indicated. However, the conservative estimates of the non-linear regression relationship for the entire sample may be desirable, to ensure adequate funding is available. Actual and estimates of the baseline source code estimator are compared in Figure 23. In general, it appears to overestimate.

Figure 24 presents the multivariate model for estimating manpower ( $R^2 = .89$ ,  $SE = 0.649$  ln units). In a worse case situation, the model will derive an estimate 9.76 times that of the preferred baseline model. Figure 25 compares the actual with the estimated manpower for the data sample. Good fit is exhibited for all but three points.

5.1.2.3 Scientific programs. Of the 27 scientific programs in the total population, six were discarded because of the associated low percent of written code delivered. These are noted in Figures 26 and 27. Ten of the 21 remaining in the sample had programs in object code of 10,000 words or greater, and six of the programs in source code were of 10,000 words or greater. The largest program in the data sample was for 58,300 object words (and 58,300 source lines).

Figure 28 presents the baseline estimating relationships for development manpower in terms of program size (in object and source code). For the total sample, the relationships are nominally linear (the exponent approximates 1.0). The higher coefficients of determination and standard

OBJECT CODE	LOG-LINEAR		NON-LINEAR	
	MM	SE	MM	SE
OVERALL	3.344	1.136	4.573	1.228
	(R <sup>2</sup> = .54, SE = 61.5)		(R <sup>2</sup> = .78, SE = 41.1)	
I ≥ 10K	4.542	1.241	10.040	1.001
	(R <sup>2</sup> = .25, SE = 80.4)		(R <sup>2</sup> = .34, SE = 75.4)	
I < 10K	4.229	0.711	3.158	0.974
	(R <sup>2</sup> = .40, SE = 11.4)		(R <sup>2</sup> = .74, SE = 7.7)	
SOURCE CODE	3.712	1.119	4.089	1.263
OVERALL	(R <sup>2</sup> = .58, SE = 59.3)		(R <sup>2</sup> = .80, SE = 41.1)	
I ≥ 10K	4.542	1.241	10.040	1.001
	(R <sup>2</sup> = .25, SE = 80.4)		(R <sup>2</sup> = .34, SE = 75.4)	
I < 10K	4.426	0.678	3.626	0.724
	(R <sup>2</sup> = .49, SE = 10.5)		(R <sup>2</sup> = .54, SE = 7.1)	


 Denotes preferred relationship

Figure 18. Baseline manpower relationships for command and control programs

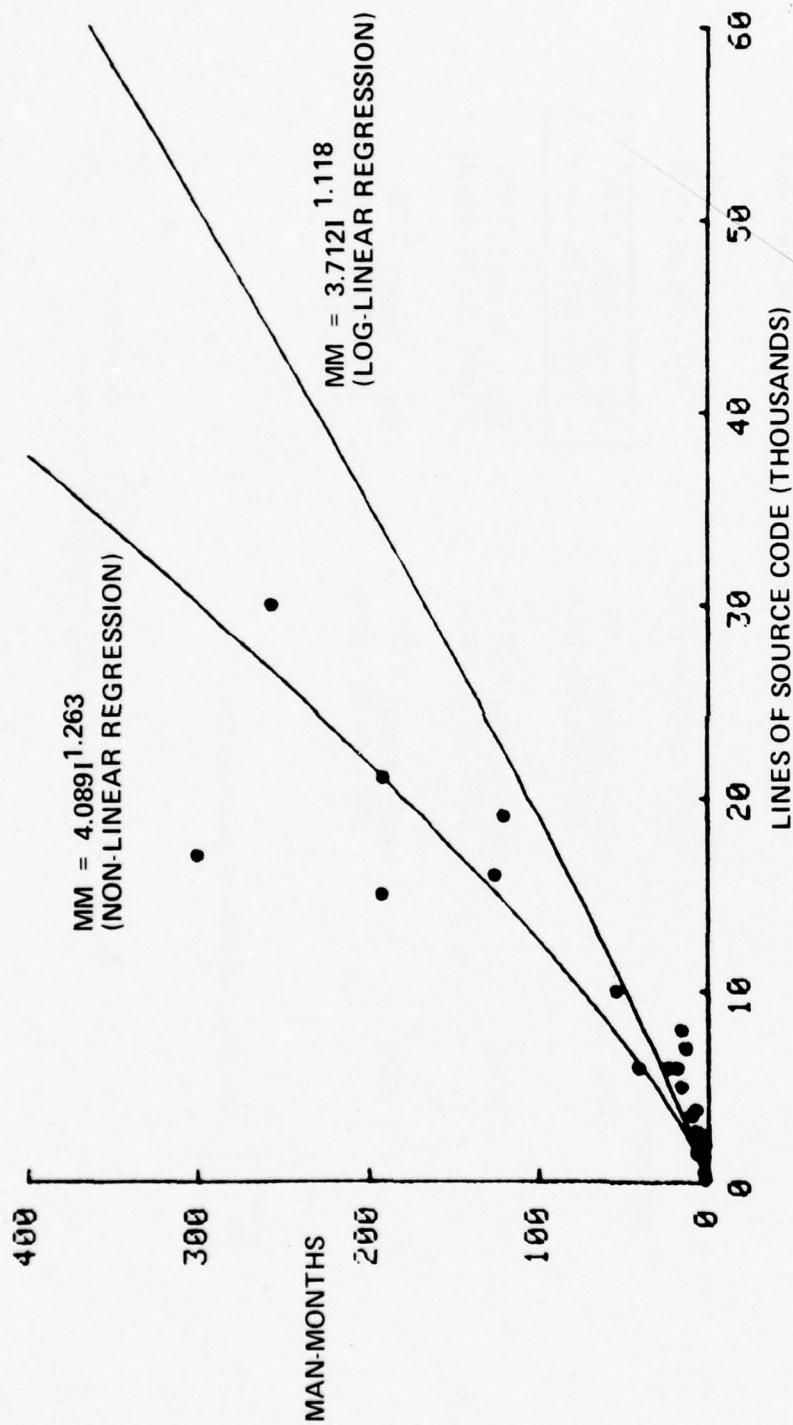


Figure 19. Relationship between program size in source code and development manpower for command and control programs

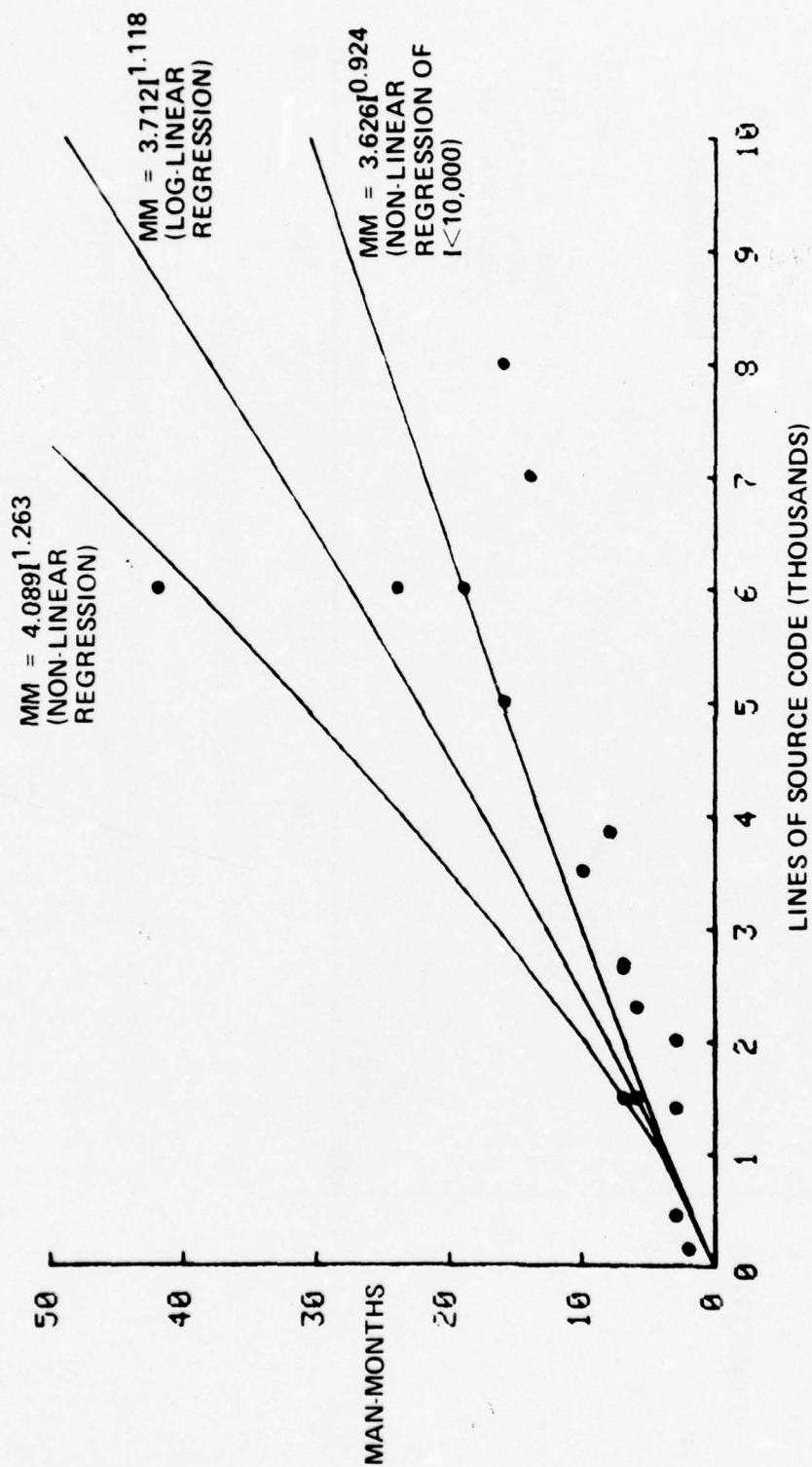


Figure 20. Relationship between program size in source code and development manpower for command and control programs ( $L < 10,000$ )

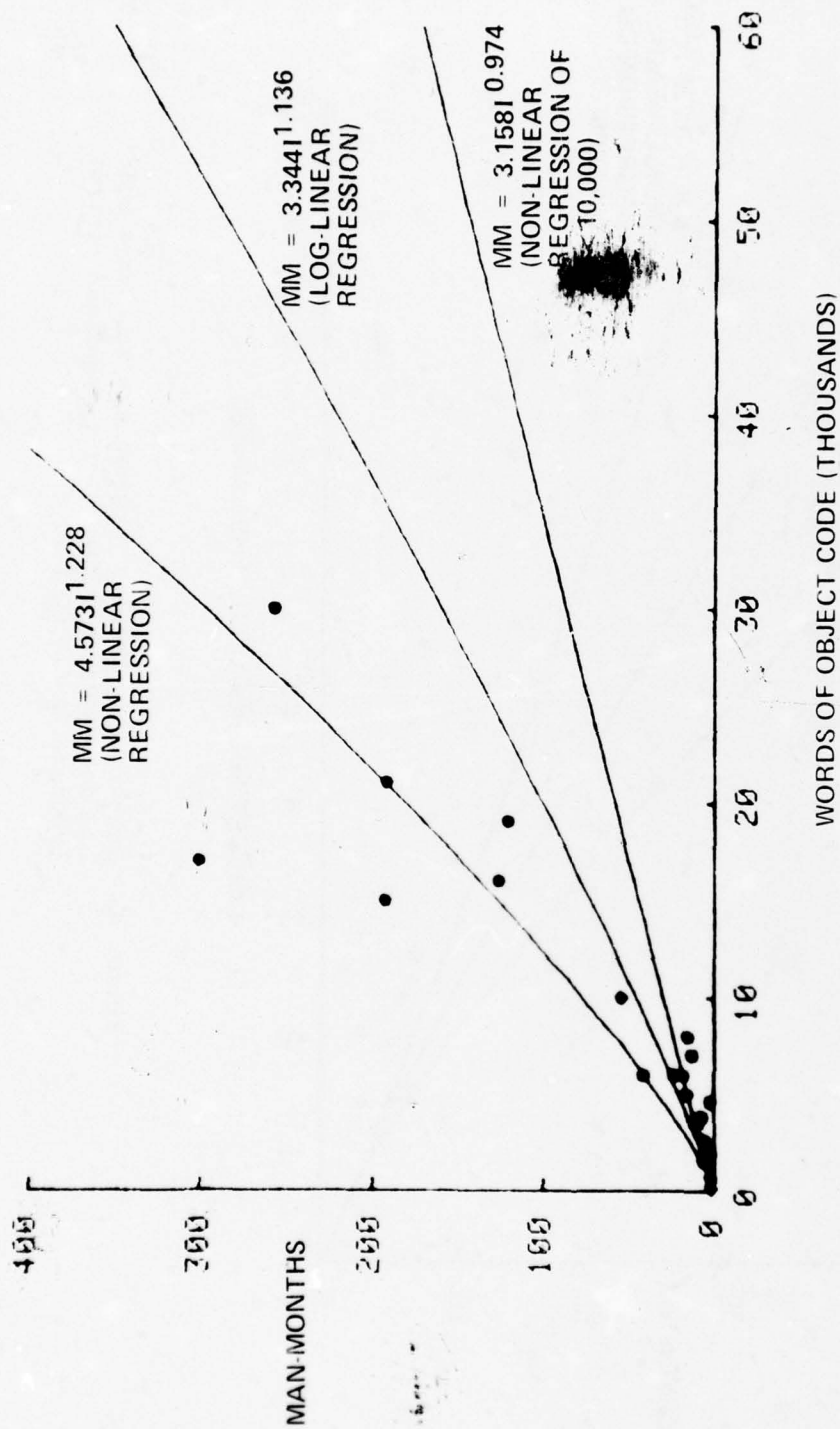


Figure 21. Relationship between program size in object code and development manpower for command and control programs

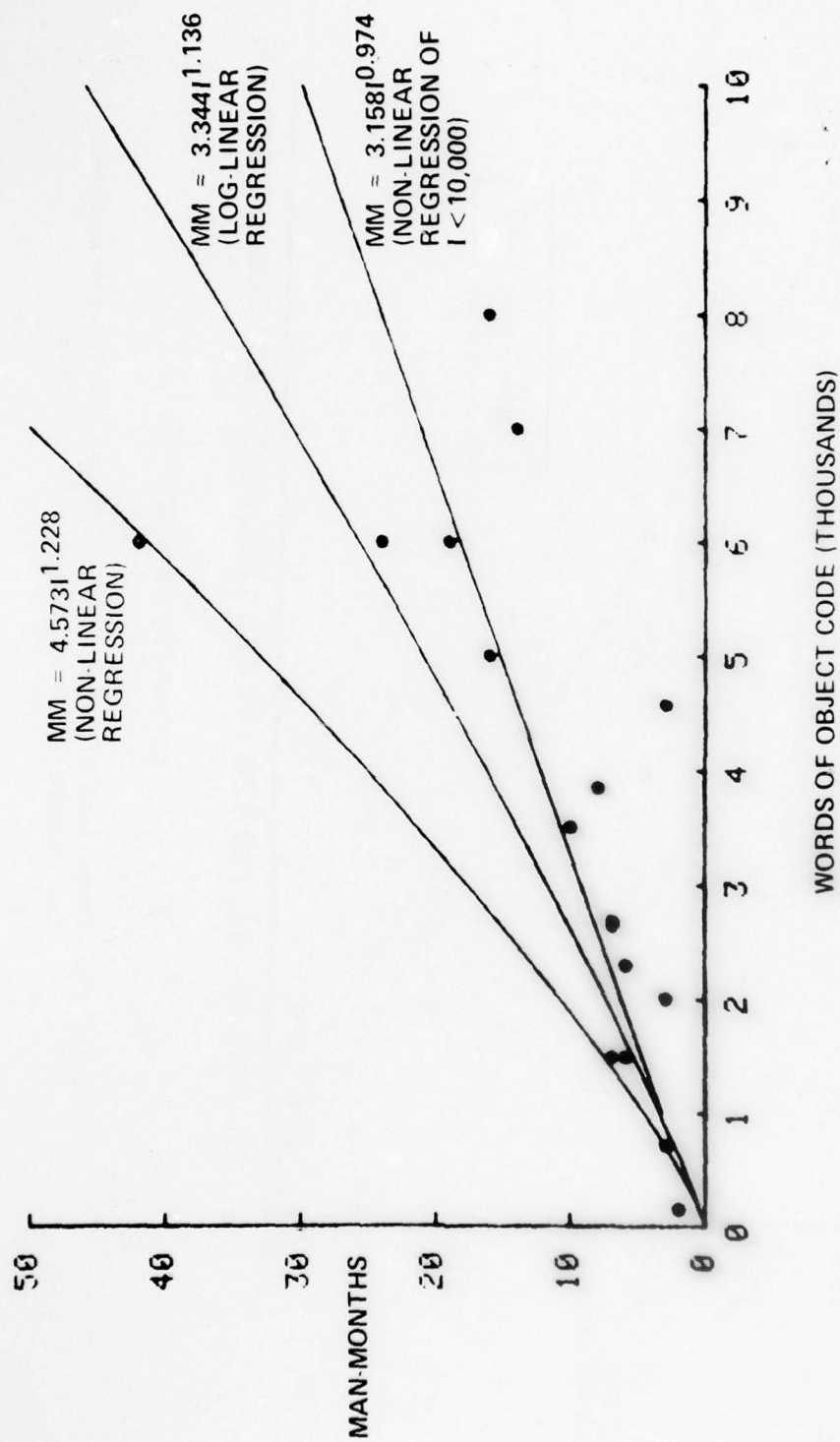


Figure 22. Relationship between program size in object code and development manpower for command and control programs (I < 10,000)

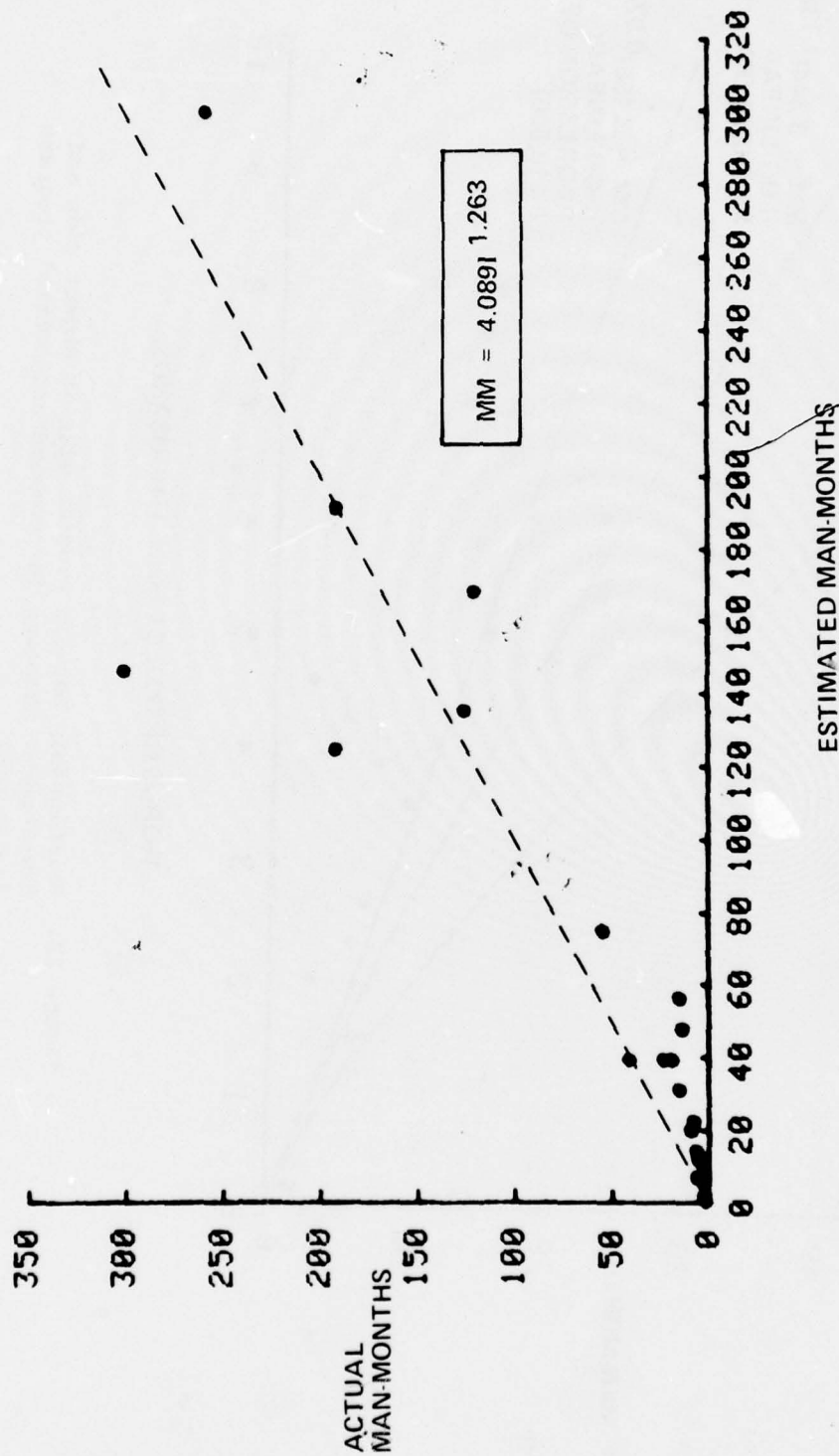


Figure 23. Comparison of actual and estimated development manpower for command and control software programs

$$MM = 0.501 \prod_{j=1}^{j=14} 1.263^{f_j}$$

FACTOR	f	YES	NO
SPECIAL DISPLAY	f <sub>1</sub>	1.11	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f <sub>2</sub>	1.00	1.54
CHANGES TO OPERATIONAL REQUIREMENTS	f <sub>3</sub>	1.05	1.00
REAL TIME OPERATION	f <sub>4</sub>	1.33	1.00
CPU MEMORY CONSTRAINT	f <sub>5</sub>	1.25	1.00
CPU TIME CONSTRAINT	f <sub>6</sub>	1.51	1.00
FIRST S/W DEVELOPED ON CPU	f <sub>7</sub>	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f <sub>8</sub>	1.67	1.00
TIME SHARING, VIS-A-VIS BATCH PROCESSING, IN DEVELOPMENT	f <sub>9</sub>	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f <sub>10</sub>	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f <sub>11</sub>	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f <sub>12</sub>	2.22	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f <sub>13</sub>	1.25	1.00
PROGRAMMER ACCESS TO COMPUTER	f <sub>14</sub>	} LIMITED: UNLIMITED:	1.00
			1.00

Figure 24. Manpower estimation model for command and control software

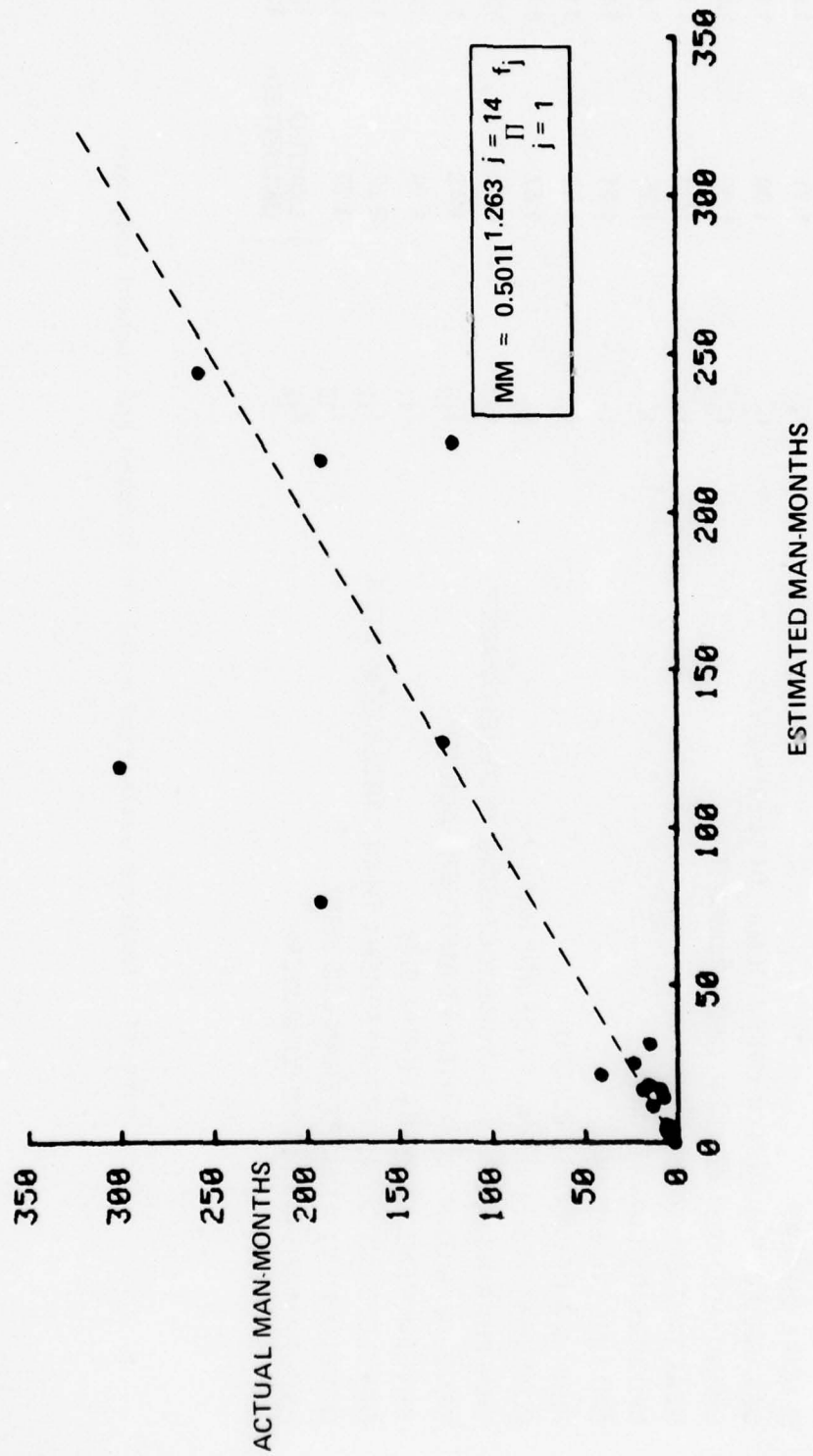


Figure 25. Comparison of actual and estimated development manpower for command and control software programs (multivariate model)

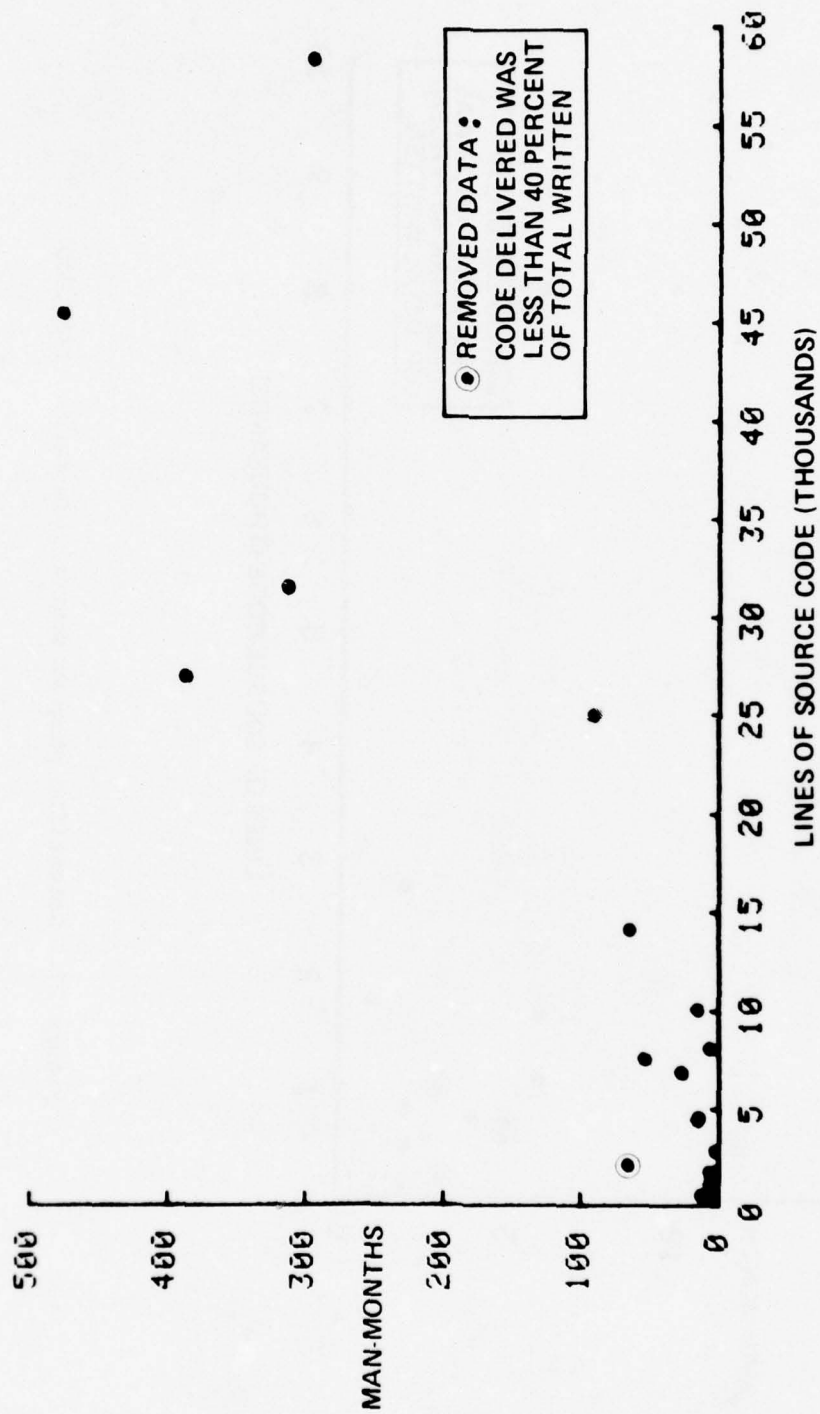


Figure 26. Scientific program source code data

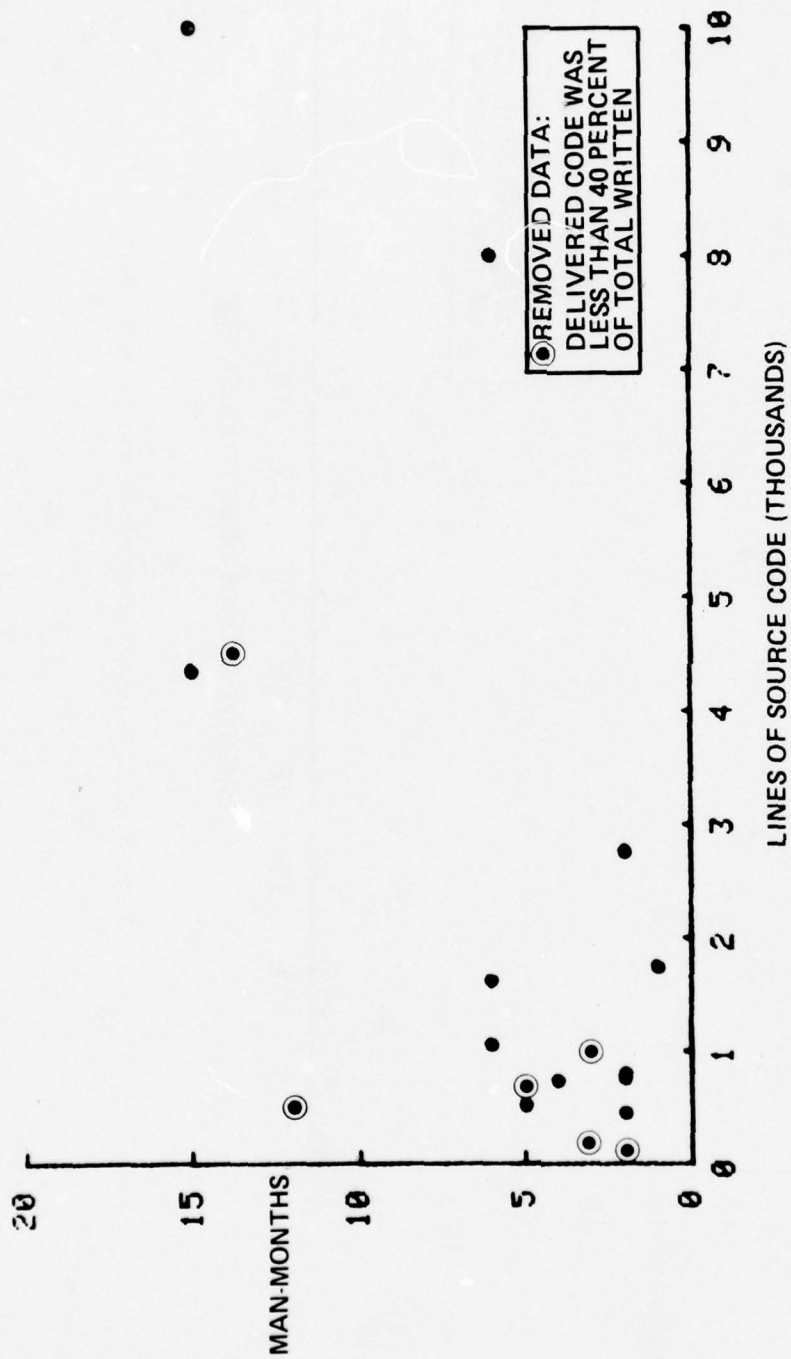


Figure 27. Scientific program source code data (I<10,000)

	LOG-LINEAR	NON-LINEAR
OBJECT CODE		
OVERALL	MM = 2.786 I 0.927 (R <sup>2</sup> = .17, SE = 137.0)	<div>MM = 4.495 I 1.068 (R<sup>2</sup> = .74, SE = 72.1)</div>
I ≥ 10K	MM = 0.935 I 1.333 (R <sup>2</sup> = .03, SE = 187.5)	MM = 6.000 I 0.996 (R <sup>2</sup> = .30, SE = 158.5)
I < 10K	MM = 3.889 I 0.120 (R <sup>2</sup> = .001, SE = 16.7)	MM = 3.927 I 0.934 (R <sup>2</sup> = .28, SE = 13.6)
SOURCE CODE		
OVERALL	MM = 3.773 I 1.047 (R <sup>2</sup> = .58, SE = 97.4)	<div>MM = 7.054 I 1.019 (R<sup>2</sup> = .78, SE = 72.1)</div>
I ≥ 10K	MM = 0.436 I 1.787 (R <sup>2</sup> = .03, SE = 191.4)	MM = 12.520 I 0.868 (R <sup>2</sup> = .53, SE = 134.5)
I < 10K	MM = 4.111 I 0.578 (R <sup>2</sup> = .22, SE = 12.8)	MM = 3.946 I 0.962 (R <sup>2</sup> = .49, SE = 11.1)



Denotes preferred relationship

Figure 28. Baseline manpower relationships for scientific programs

errors of the non-linear regression equations make this set more acceptable. This sample was also structured into subsets of larger and smaller programs. The estimators for these subsets are not considered acceptable.

The fits to the highly dispersed data, indicated in Figures 29 through 32, are not good. Because of the scatter, the more conservative estimates for the non-linear regression baseline estimator are preferred. Figure 33 compares the estimated and actual manpower for the data using the non-linear regression baseline equation; the relationship tends to overestimate. Only three of the 21 in the sample were underestimated significantly.

Figure 34 presents the multivariate model ( $R^2 = .90$ ,  $SE = 0.964$  ln units). In a worse case situation, this model will generate an estimate 38.13 times the estimate of the baseline model. A comparison of actual and estimated manpower for the data in the sample indicated that this estimator overestimates very significantly for large programs. For small programs ( $I < 10,000$ ), it estimates with about the same accuracy as the non-linear relationship.

5.1.2.4 Business programs. This category of software represented the largest subsample of the population analyzed. It included 58 sets of data; 21 were deleted because they reflected productivity of greater than 2000 source lines per man-month (too high to be considered valid) or the delivered code was less than 40 percent of the total written. Figures 35 and 36 indicate the data deleted from the sample. In source code, four of the 58 involved software with greater than 10,000 lines; ten of the 58 had object words numbering greater than 10,000. The largest programs in the subsample were 35,000 lines of source code and 46,800 words of object code.

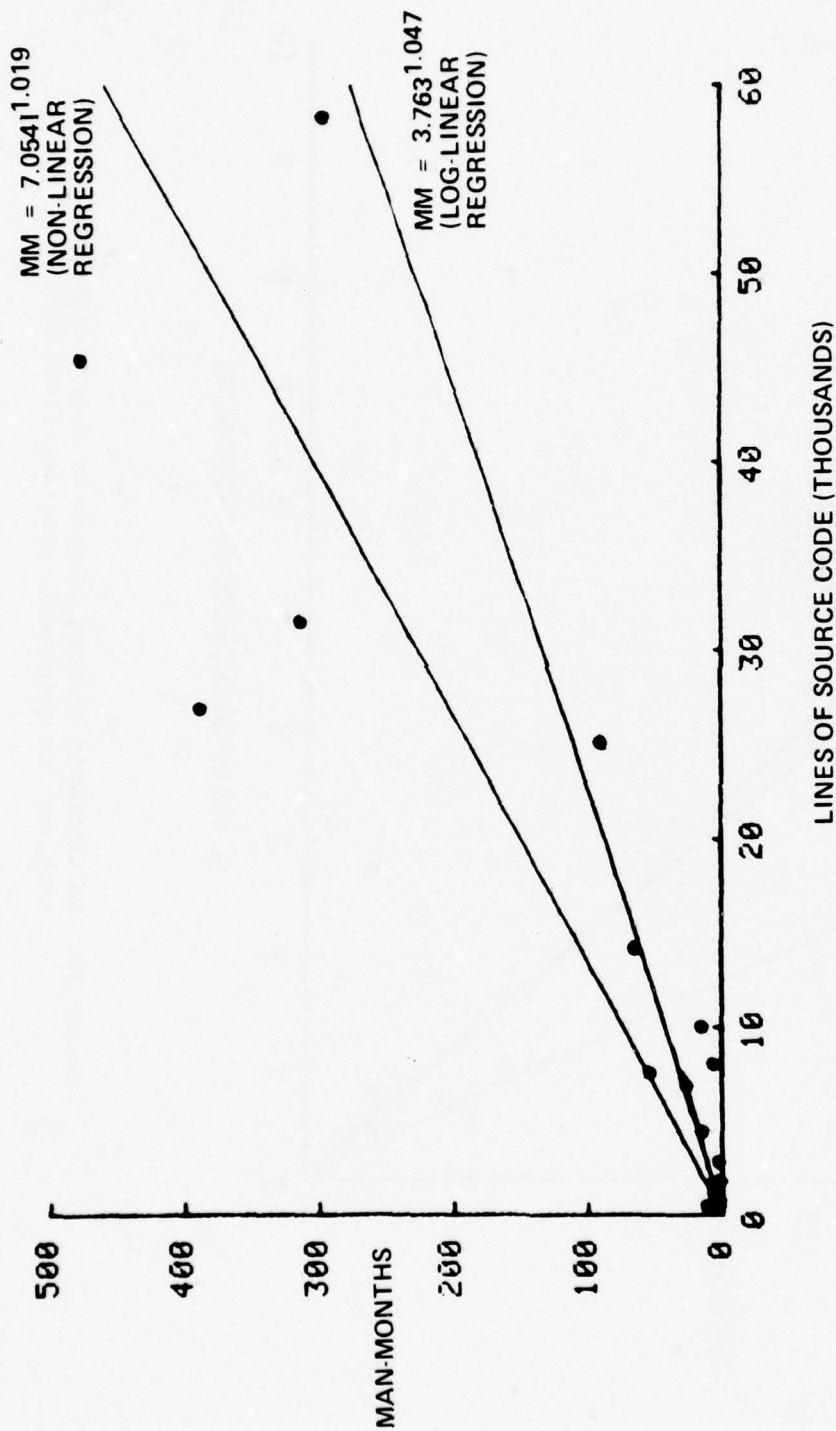


Figure 29. Relationship between program in source code and manpower for scientific programs

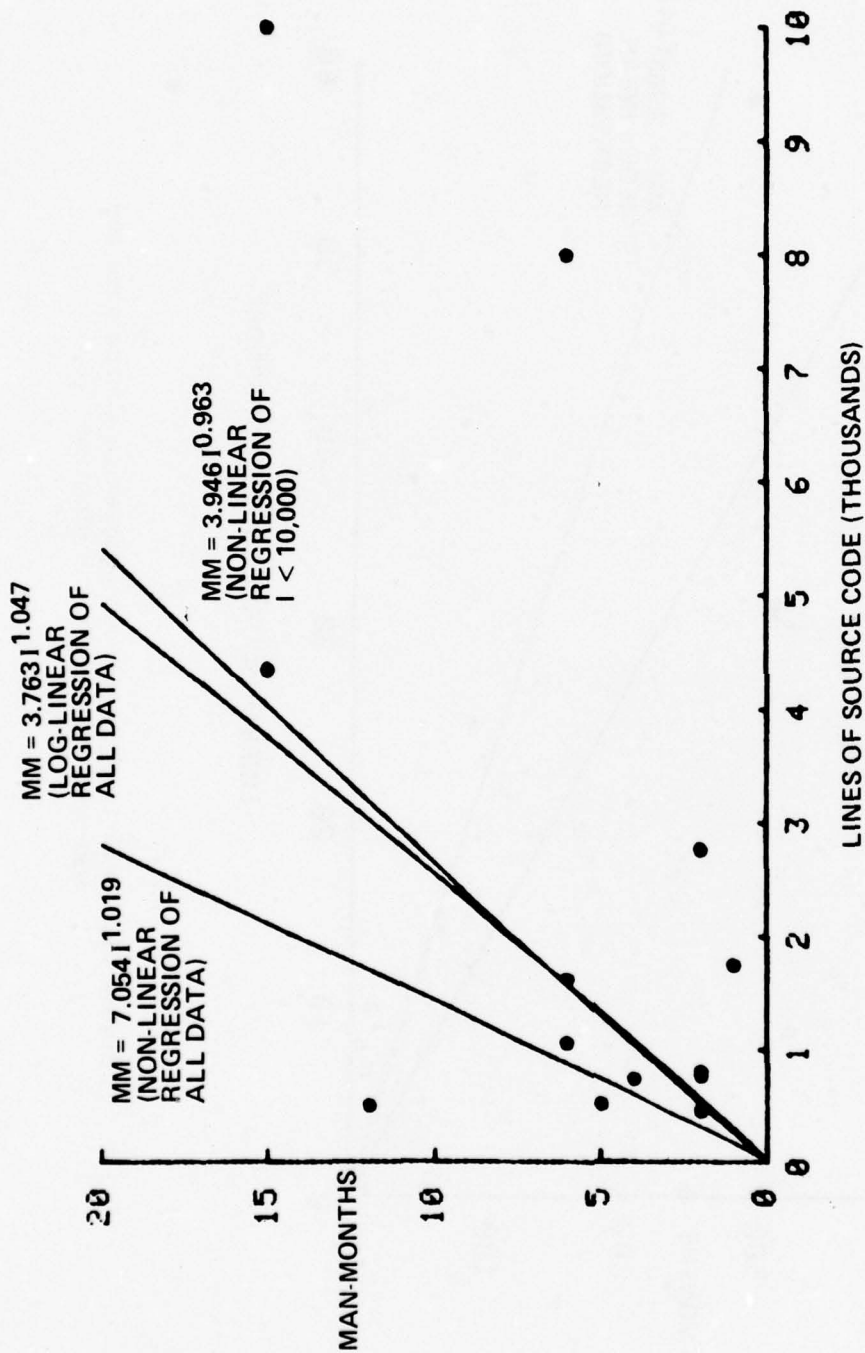


Figure 30. Relationship between program in source code and manpower for scientific programs ( $I < 10,000$ )

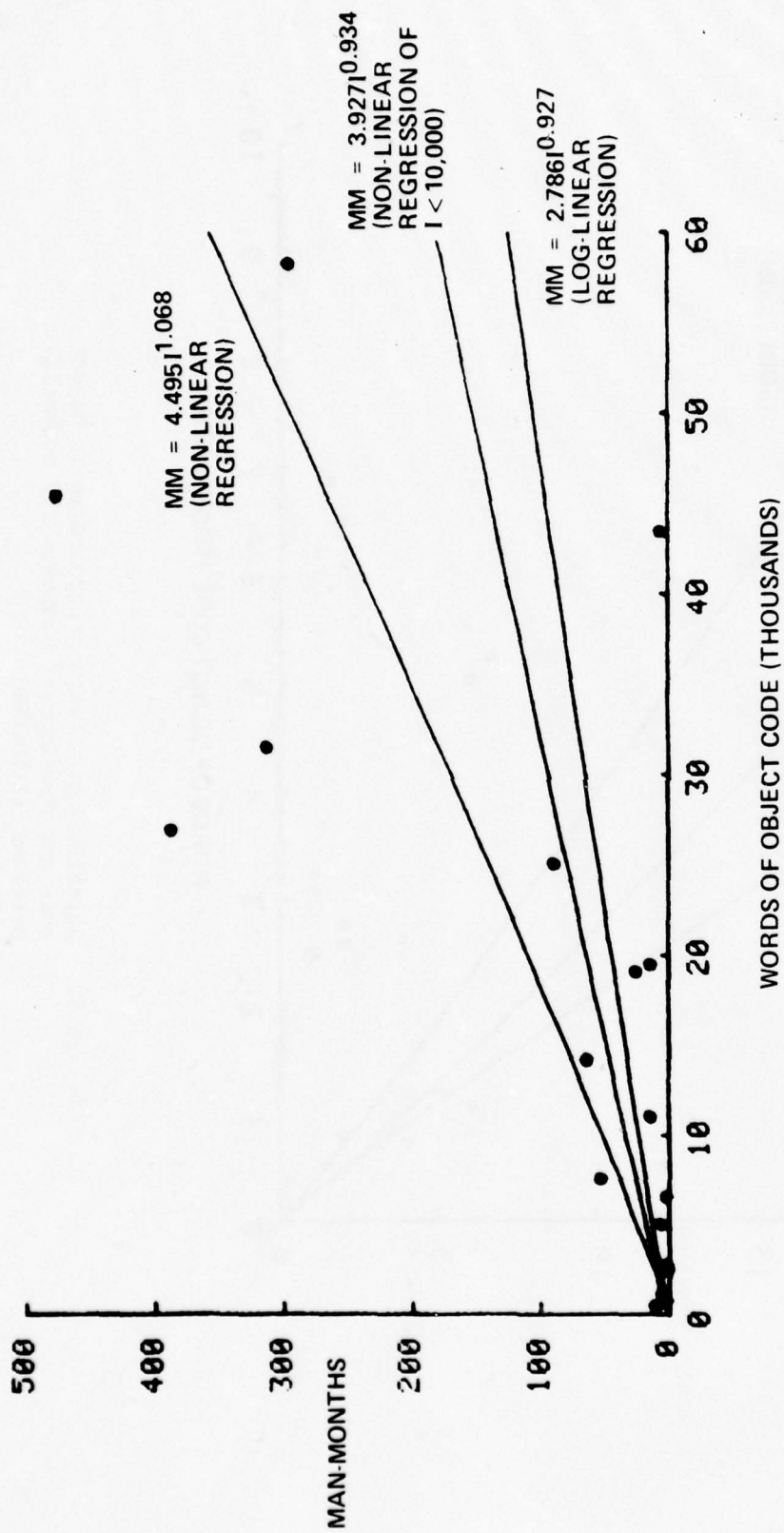


Figure 31. Relationship between program size in object code and development manpower for scientific programs

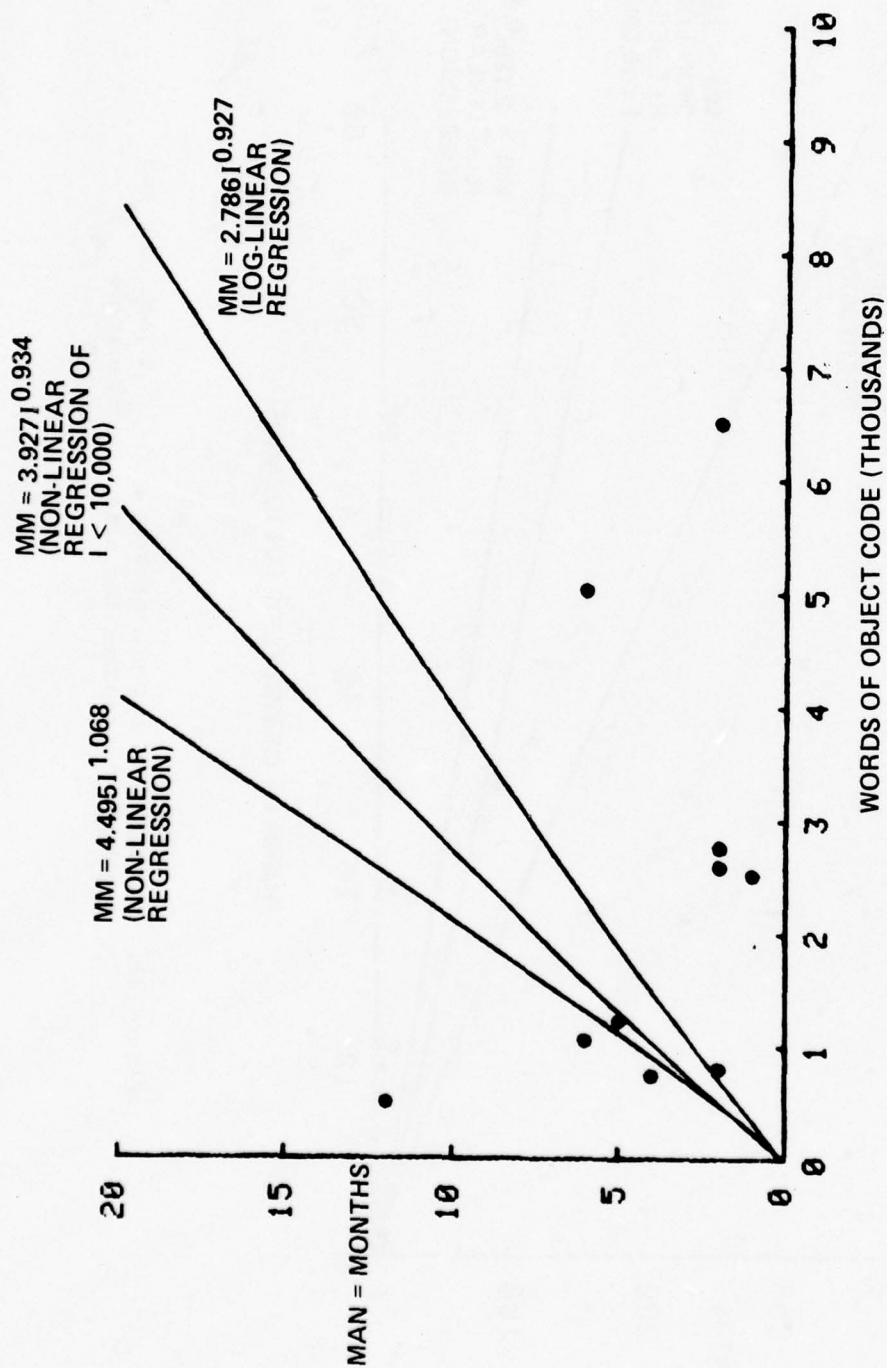


Figure 32. Relationship between program size in object code and development manpower for scientific programs ( $I < 10,000$ )

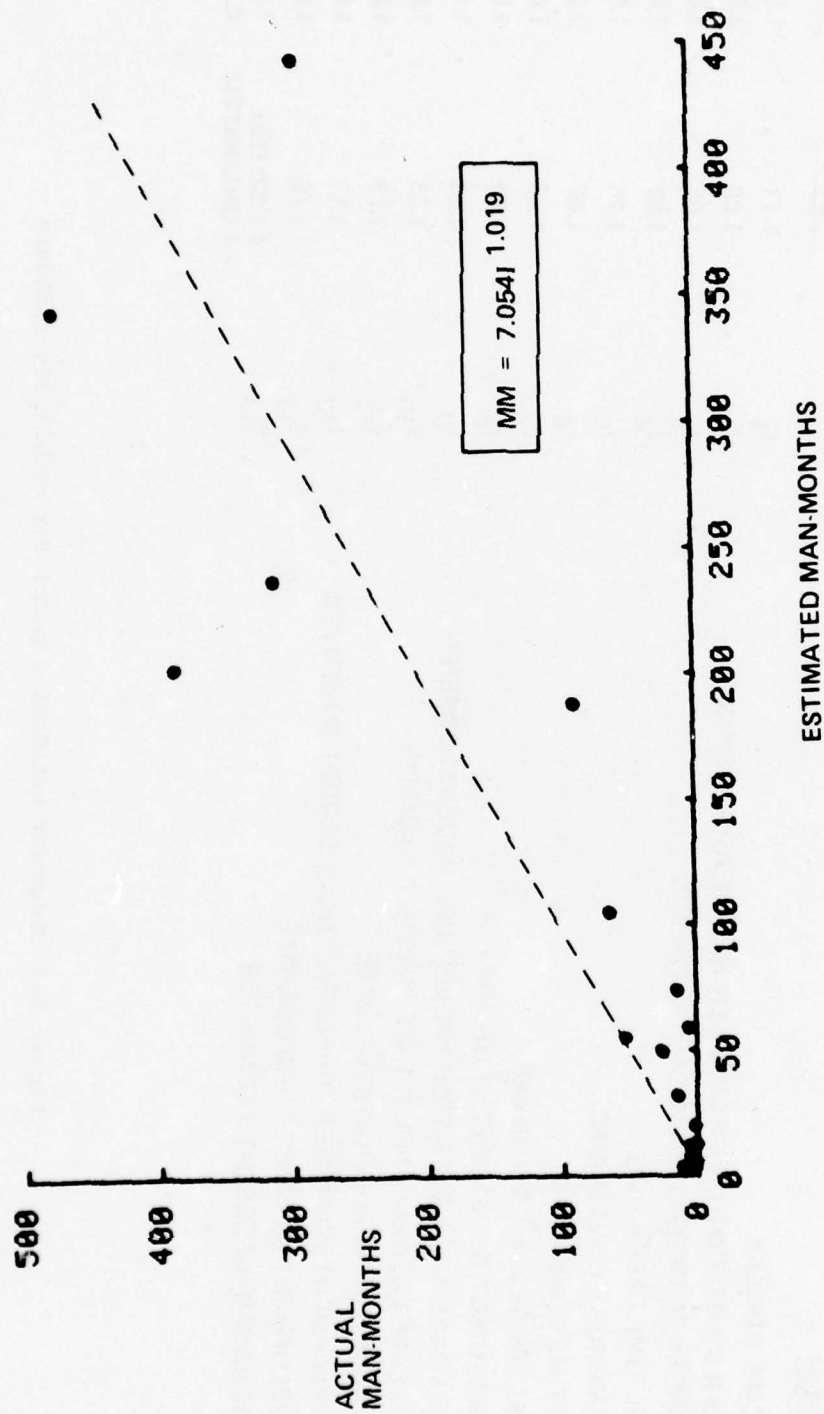


Figure 33. Comparison of actual and estimated development manpower for scientific software programs

$$MM = 2.011 \prod_{j=1}^{j=14} f_j$$

FACTOR	f	YES	NO
SPECIAL DISPLAY	f <sub>1</sub>	1.11	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f <sub>2</sub>	1.00	2.00
CHANGES TO OPERATIONAL REQUIREMENTS	f <sub>3</sub>	1.05	1.00
REAL TIME OPERATION	f <sub>4</sub>	1.67	1.00
CPU MEMORY CONSTRAINT	f <sub>5</sub>	1.25	1.00
CPU TIME CONSTRAINT	f <sub>6</sub>	1.67	1.00
FIRST S/W DEVELOPED ON CPU	f <sub>7</sub>	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f <sub>8</sub>	2.22	1.00
TIME SHARE, VIS-A-VIS BATCH PROCESSING, IN DEVELOPMENT	f <sub>9</sub>	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f <sub>10</sub>	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f <sub>11</sub>	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f <sub>12</sub>	1.11	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f <sub>13</sub>	1.75	1.00
PROGRAMMER ACCESS TO COMPUTER	f <sub>14</sub>		
		} LIMITED: 1.00	1.00
		} UNLIMITED: 0.67	0.67

Figure 34. Manpower estimation model for scientific software

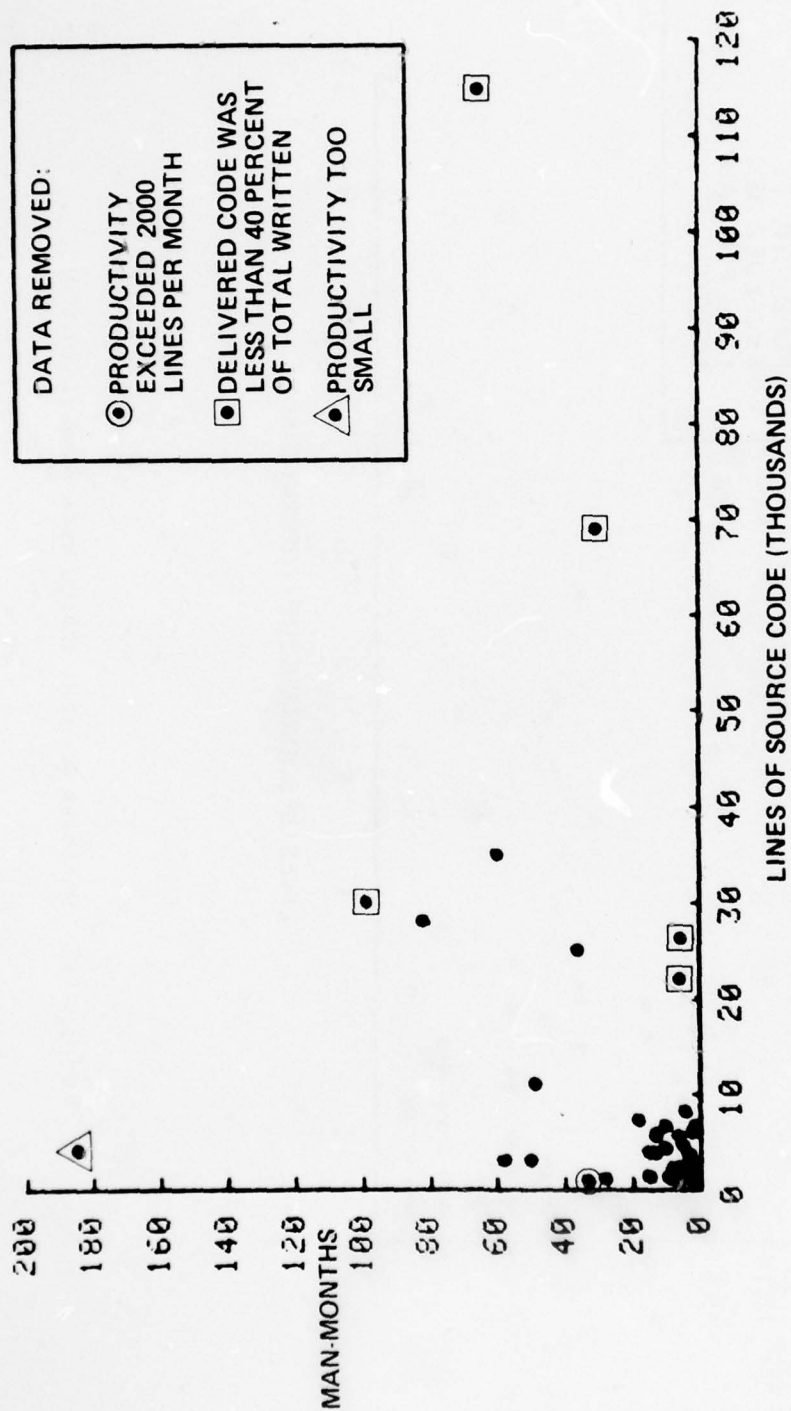


Figure 35. Business program source code data



Figure 37 presents the baseline estimators for business programs derived using log-linear and non-linear regression. All have exponents less than one, indicating apparent economies of scale. Thus, the hypothesis of the General Research Corporation, [59], indicating that economies of scale exist in software development, may be valid for business programs. Since a very few of the programs exceed 10,000 instructions, subsets of the sample were not analyzed. The fits of the relationships to the data are indicated in Figures 38 through 41. Because the fit to the data of the non-linear regression relationships, in general, appears to be better, and since the estimates from these relationships are more conservative, they are recommended for use in estimating the manpower required for the development of business software programs.

Figure 42 compares actual and estimated manpower for the business program data. Several of the highly dispersed data are underestimated. Therefore, preference should be given to the multivariate estimator presented in Figure 43 ( $R^2 = .783$ ,  $SE = 0.588$  ln units) which appears to estimate higher costs for larger programs in the data sample. For a worse case situation, the multivariate relationship will develop an estimate 7.93 times that of the baseline relationship.

5.1.2.5 Utility programs. Twenty-six data sets remained in the sample after removal of two because they did not satisfy the 40 percent criteria for deliverable code. The utility programs are the only subsample of the total data sample in which a majority of the programs exceeded 10,000 words of object code (19 of 26) and 10,000 words of source code (16 of 26). The data, presented in Figure 44, is highly dispersed. An attempt was made to stratify the data to determine if it reflected various subsets. However, the only subset that could be established is that of time constraint which, in general, reflected significantly higher development manpower.

The baseline relationships are presented in Figure 45. The coefficients of determination are very low because of the highly dispersed data. Of significance is that the exponent, contrary to expectations, is less than 1.0.

# LOG-LINEAR

# NON-LINEAR

OBJECT CODE

MM = 3.0031 0.481  
(R2 = .17, SE = 15.6)

MM = 2.8951 0.784  
(R2 = .48, SE = 12.4)

SOURCE CODE

MM = 3.9751 0.733  
(R2 = .55, SE = 11.4)

MM = 4.4951 0.781  
(R2 = .61, SE = 10.7)



Denotes preferred relationship

Figure 37. Baseline manpower relationships for business programs

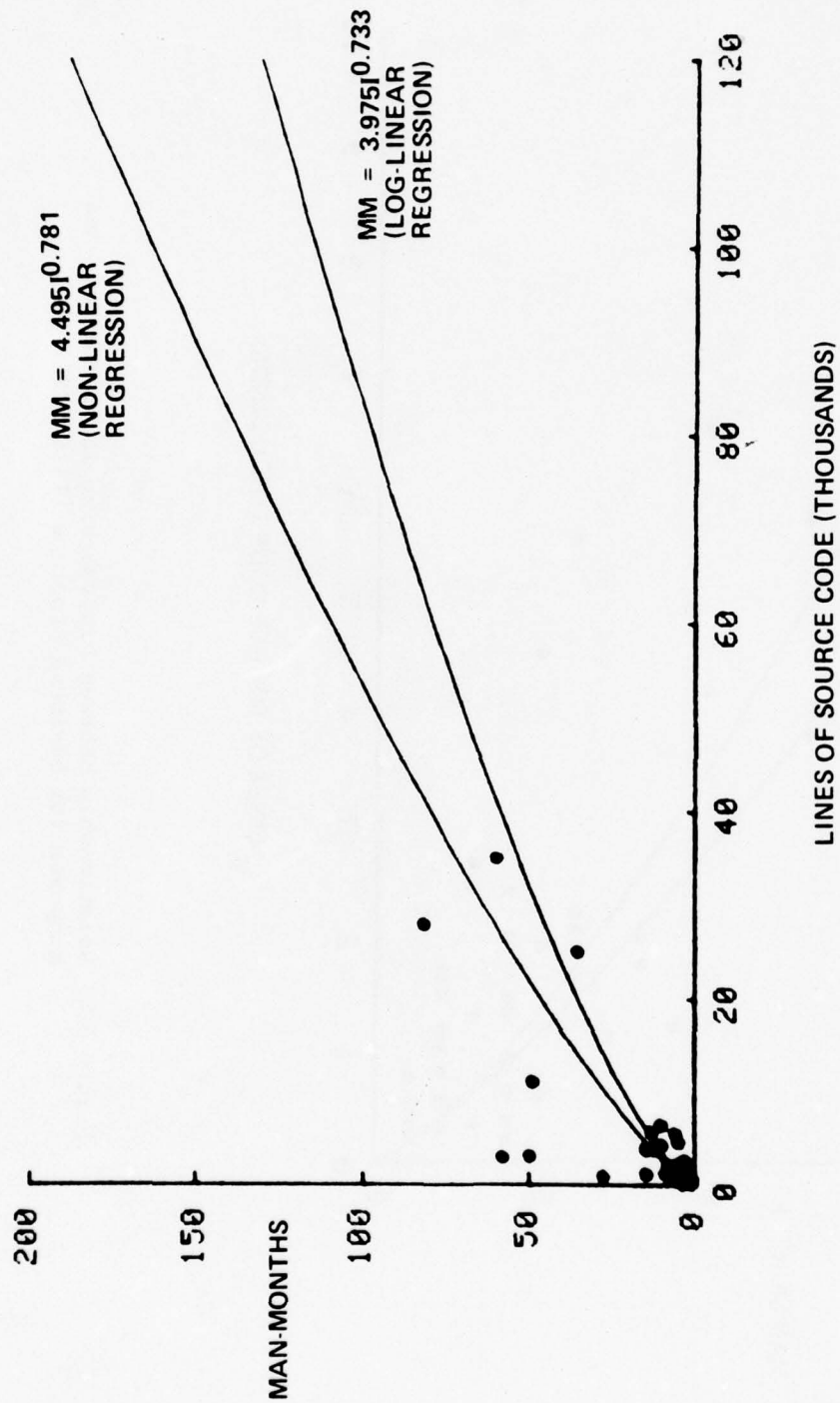


Figure 38. Relationship between program size in source code and manpower for business programs

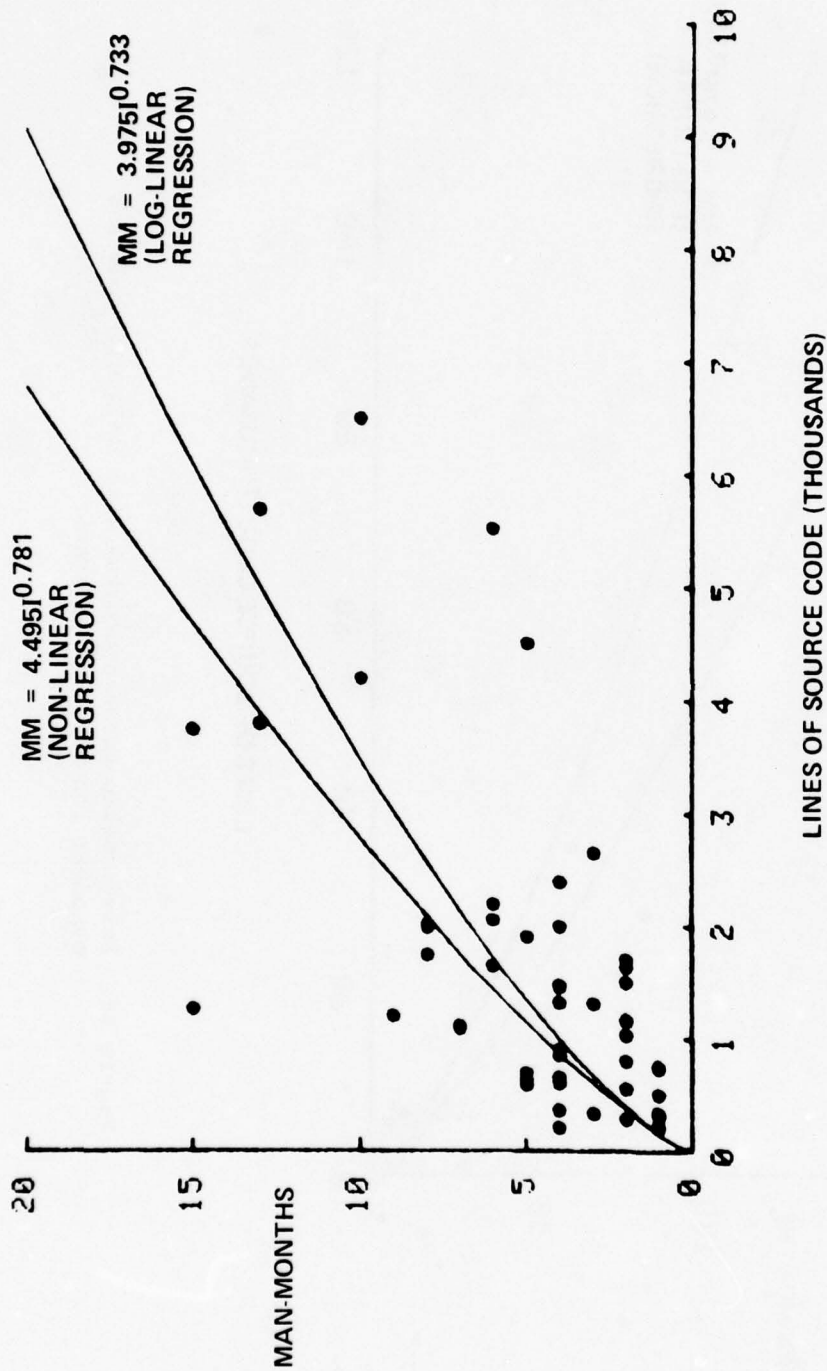


Figure 39. Relationship between program size in source code and manpower for business programs ( $I < 10,000$ )

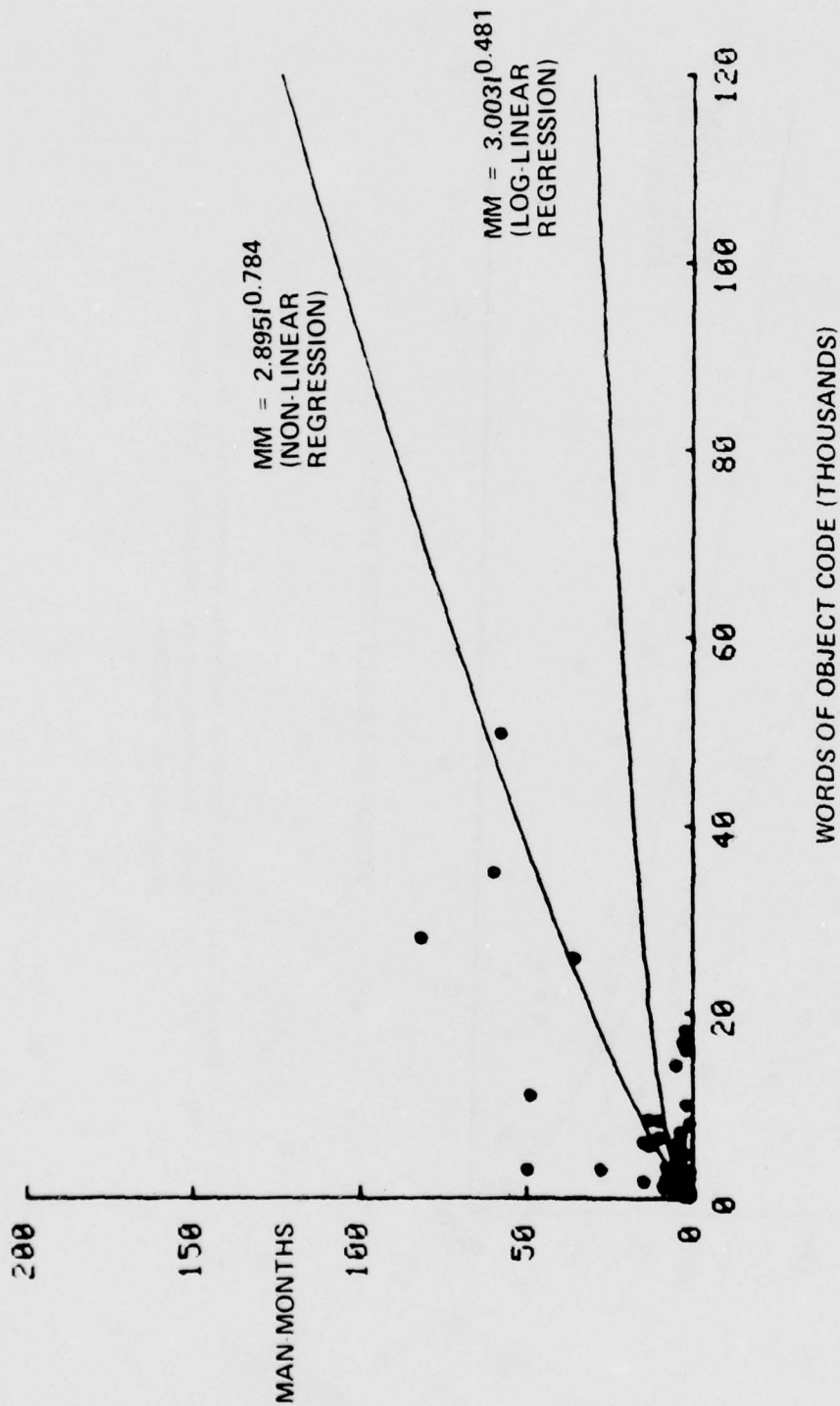


Figure 40. Relationship between program size in object code and development manpower for business programs

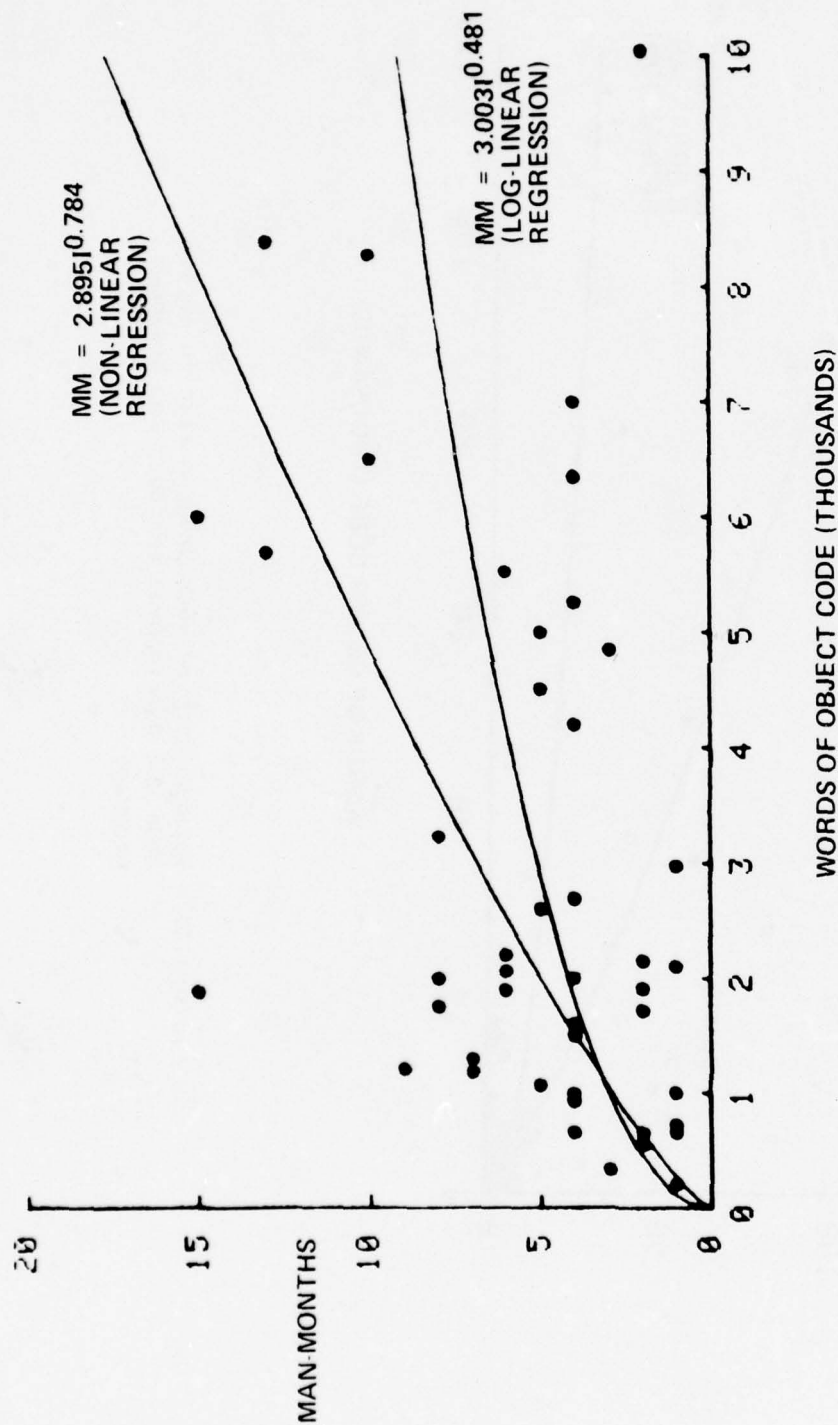


Figure 41. Relationship between program size in object code and development manpower for business programs ( $I < 10,000$ )

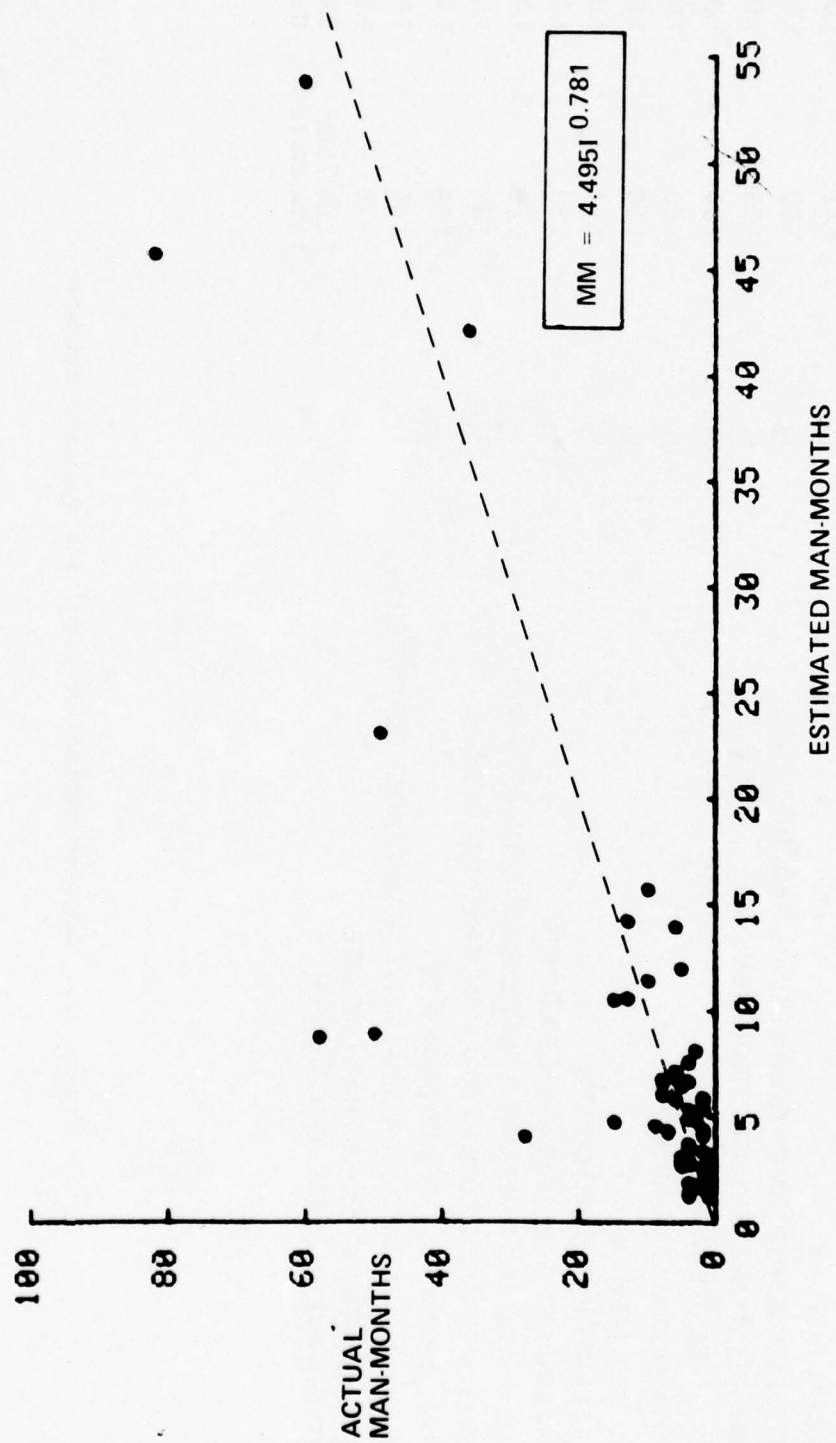


Figure 42. Comparison of actual and estimated development manpower for business software programs

$$MM = 3.7421 \sum_{j=1}^{j=14} 0.781 f_j$$

FACTOR	f	YES	NO
SPECIAL DISPLAY	f <sub>1</sub>	1.43	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f <sub>2</sub>	1.00	1.00
CHANGES TO OPERATIONAL REQUIREMENTS	f <sub>3</sub>	1.05	1.00
REAL TIME OPERATION	f <sub>4</sub>	1.00	1.00
CPU MEMORY CONSTRAINT	f <sub>5</sub>	1.00	1.00
CPU TIME CONSTRAINT	f <sub>6</sub>	1.00	1.00
FIRST S/W DEVELOPED ON CPU	f <sub>7</sub>	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f <sub>8</sub>	1.33	1.00
TIME SHARE, VIS-A-VIS BATCH PROCESSING, IN DEVELOPMENT	f <sub>9</sub>	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f <sub>10</sub>	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f <sub>11</sub>	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f <sub>12</sub>	1.00	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f <sub>13</sub>	1.25	1.00
PROGRAMMER ACCESS TO COMPUTER	f <sub>14</sub>	{ LIMITED: UNLIMITED:	{ 1.00 0.90

Figure 43. Manpower estimation model for business software

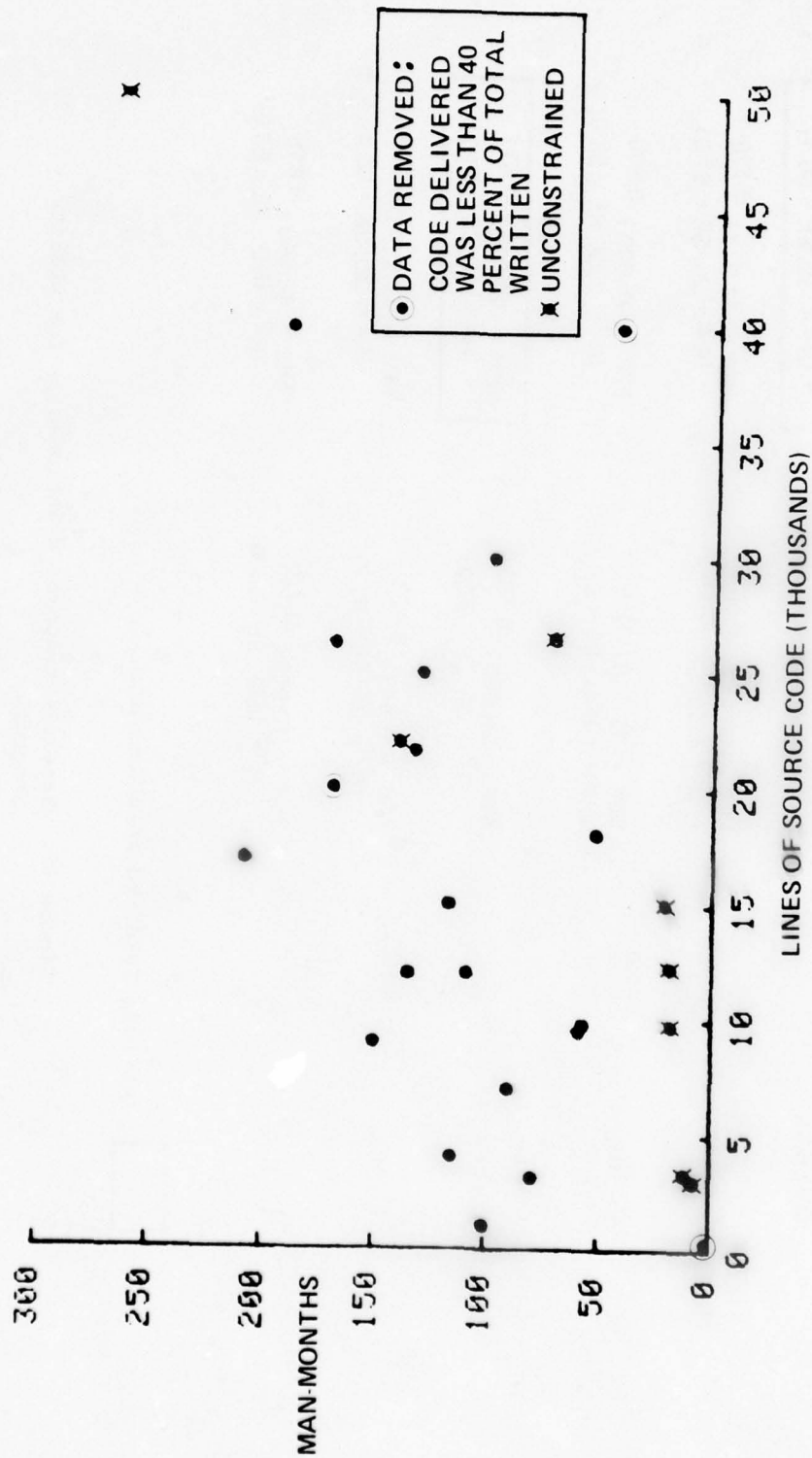


Figure 44. Utility program source code data

		LOG-LINEAR	NON-LINEAR
OBJECT CODE	OVERALL	MM = 12.1421 0.654 (R <sup>2</sup> = .21, SE = 61.4)	MM = 12.0391 0.719 (R <sup>2</sup> = .30, SE = 58.1)
	I ≥ 10K	MM = 16.6171 0.561 (R <sup>2</sup> = .13, SE = 64.9)	MM = 11.9141 0.726 (R <sup>2</sup> = .23, SE = 61.0)
	I < 10K	MM = 12.5661 0.617 (R <sup>2</sup> = IND., SE = 57.7)	MM = 15.0001 0.800 (R <sup>2</sup> = IND., SE = 58.0)
	SOURCE CODE		
	OVERALL	MM = 10.8661 0.753 (R <sup>2</sup> = .42, SE = 52.9)	MM = 10.0781 0.811 (R <sup>2</sup> = .45, SE = 51.7)
	I ≥ 10K	MM = 6.3431 0.922 (R <sup>2</sup> = .32, SE = 56.7)	MM = 11.9271 0.762 (R <sup>2</sup> = .33, SE = 56.4)
	I < 10K	MM = 10.9651 0.779 (R <sup>2</sup> = IND., SE = 52.5)	MM = 10.0081 0.815 (R <sup>2</sup> = IND., SE = 53.0)
		<div style="border: 1px solid black; width: 30px; height: 30px; display: inline-block; vertical-align: middle;"></div> Denotes preferred relationships	

Figure 45. Baseline manpower relationships for utility programs

However, when the regression was performed on the unconstrained and constrained data subsets, the following relationships were obtained:

- Unconstrained

$$MM = 1.426I^{1.243} \quad (\text{Equation 5})$$
$$(R^2 = .77, SE = 48.9)$$

where MM is the manpower, in man-months

I is the program size, in thousands of source lines

- Constrained

$$MM = 40.69I^{0.357} \quad (\text{Equation 6})$$
$$(R^2 = .16, SE = 47.4)$$

Figures 46 and 47 indicate the fit of the baseline relationships to the data. The non-linear regression models give a better fit than do the linear regression models. Figure 48 compares actual and estimated manpower for the data in the sample using the non-linear regression model. In spite of its conservatism, half of the data points are underestimated by the relationship. Except for the time unconstrained relationship, the adequacy of the estimators is suspect. For estimating purposes it is recommended that the baseline relationship in Figure 45 be used unless the program is known not to be time constrained. Then, it is suggested that Equation 5 be used.

The multivariate model is presented in Figure 49 ( $R^2 = .61$ ,  $SE = 0.817$  ln units). The lack of an impact for special displays or for detailed definition of operational requirements is unexpected. As variables were entered or deleted from the interactive regression analysis, little impact was indicated on these variables. However, it is recommended that a factor of 1.10 be used, appropriately as noted, in lieu of 1.00. In a worse case, the multivariate relationship will exceed the estimate of

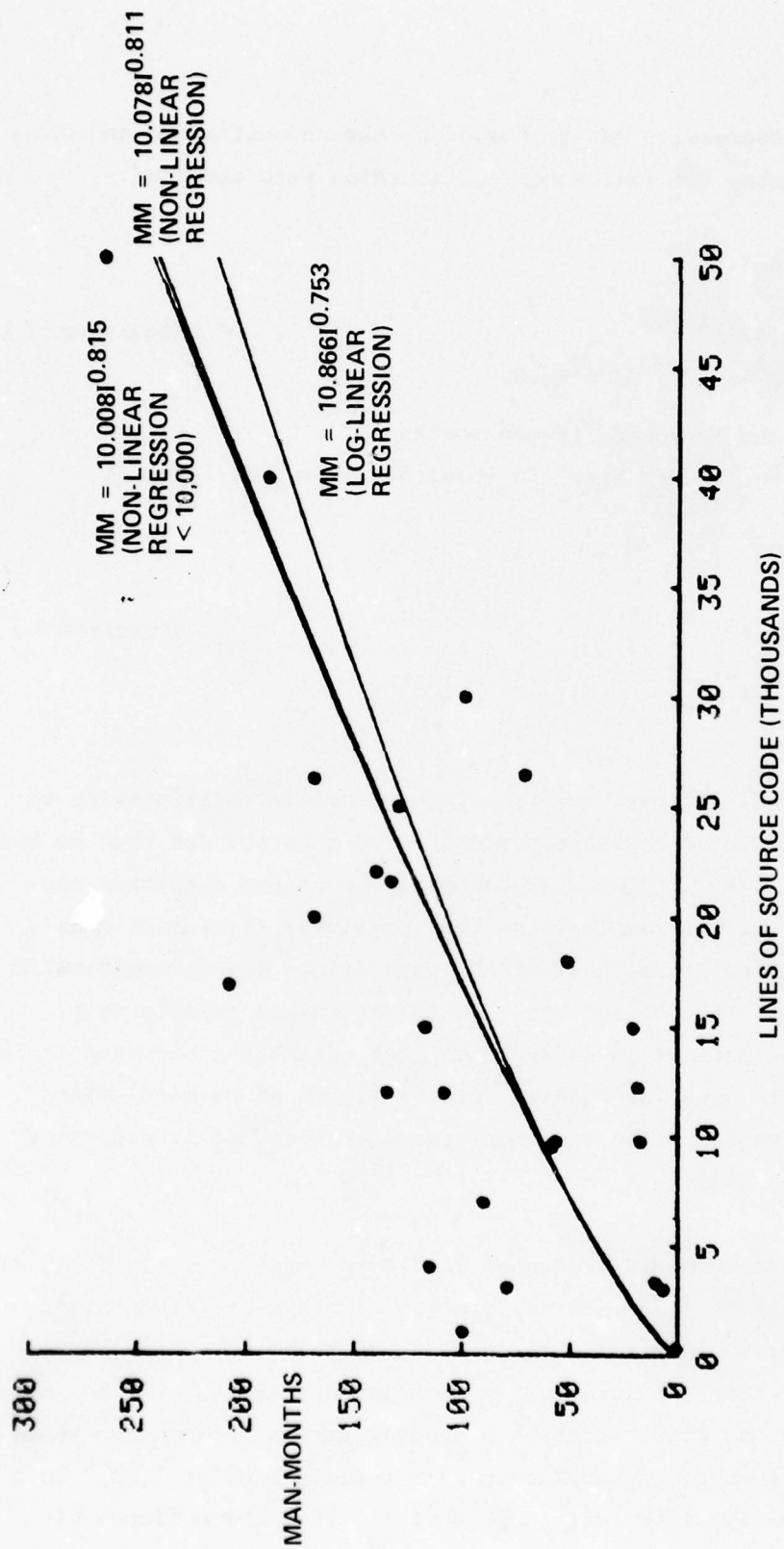


Figure 46. Relationship between program size in source code and development manpower for utility programs

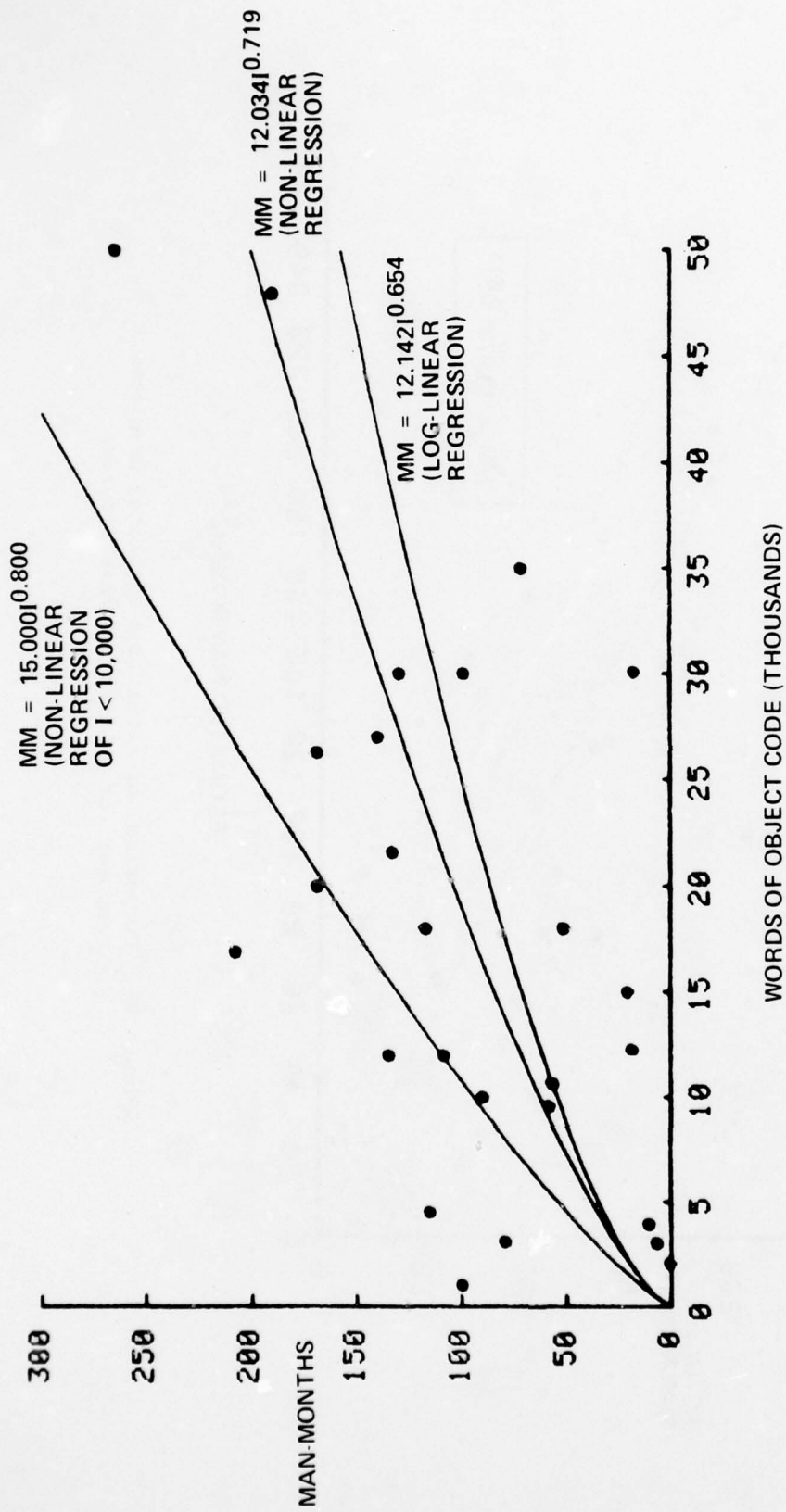


Figure 47. Relationship between program size in object code and development manpower for utility programs

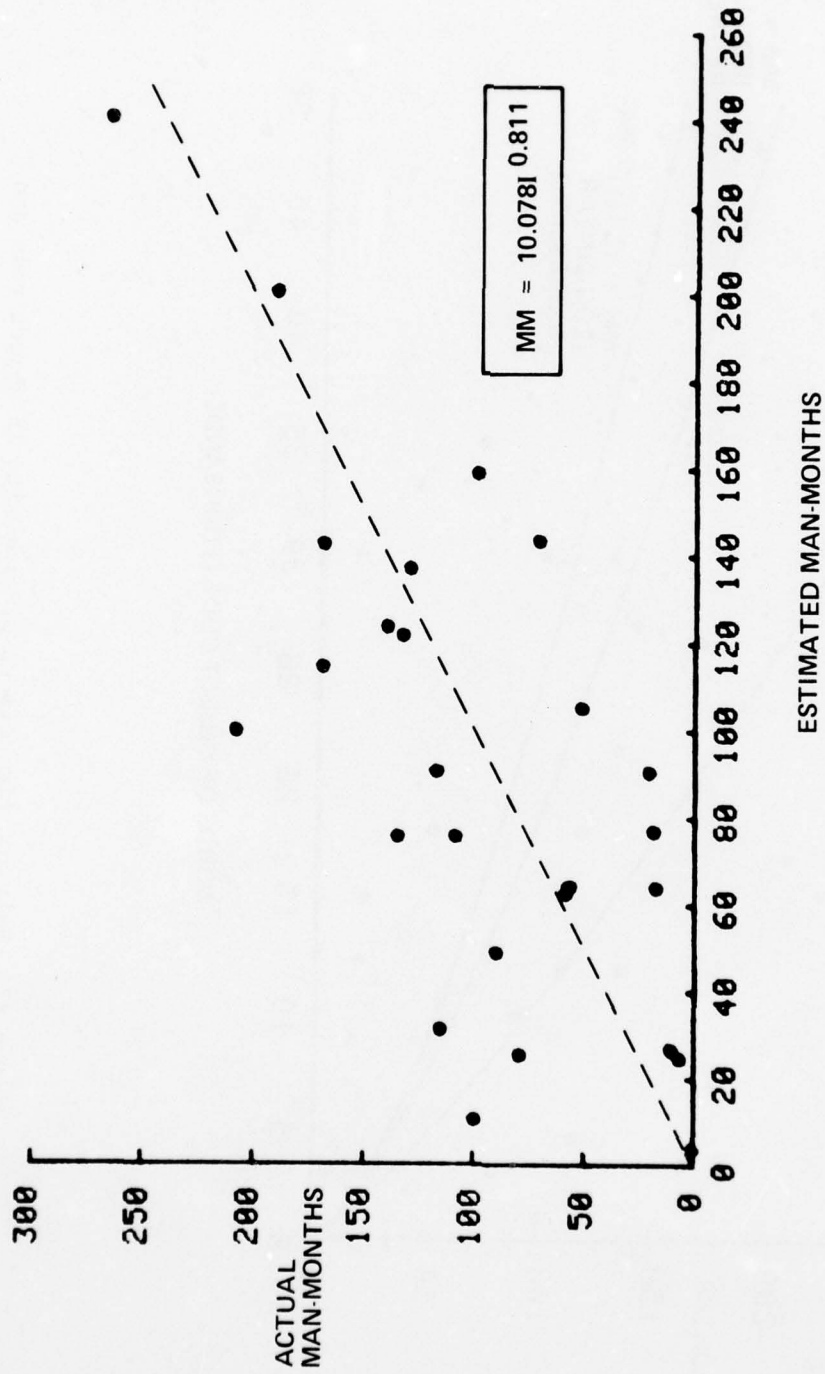


Figure 48. Comparison of actual and estimated development manpower for utility software programs

$$MM = 1.744 \prod_{j=1}^{j=14} f_j$$

FACTOR	f	YES	NO
SPECIAL DISPLAY	f <sub>1</sub>	1.00 *	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f <sub>2</sub>	1.00	1.00 *
CHANGES TO OPERATIONAL REQUIREMENTS	f <sub>3</sub>	1.05	1.00
REAL TIME OPERATION	f <sub>4</sub>	1.43	1.00
CPU MEMORY CONSTRAINT	f <sub>5</sub>	1.18	1.00
CPU TIME CONSTRAINT	f <sub>6</sub>	2.32	1.00
FIRST S/W DEVELOPED ON CPU	f <sub>7</sub>	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f <sub>8</sub>	1.25	1.00
TIME SHARE, VIS-A-VIS BATCH PROCESSING, IN DEVELOPMENT	f <sub>9</sub>	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f <sub>10</sub>	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f <sub>11</sub>	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f <sub>12</sub>	1.43	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f <sub>13</sub>	1.21	1.00
PROGRAMMER ACCESS TO COMPUTER	f <sub>14</sub>	{ LIMITED: UNLIMITED:	1.00
			0.67

\* SUGGEST 1.10

Figure 49. Manpower estimation model for utility software

the non-linear baseline by a factor of 7.10 (taking into consideration the higher factors for special display and detailed definition of requirements). As with the other applications, this relationship appears to overestimate the resource requirements. However, the number of times it underestimated the requirements was very few; therefore, this relationship would be desirable for budgeting purposes. When estimating actual costs, the baseline relationships should be used (except when the development is known to be unconstrained in response time, then equation 5 should be used).

5.1.3 Software sizing. Most of the error in estimating costs of software development is attributed to error in estimating the size of the program. Too often, size projections are made with little or no design analysis. Experience has shown that major software developers can underestimate size by a factor of three if insufficient attention is given to analyzing the software requirements. Unfortunately, if development contracts are awarded on the basis of cursory projections, drastic cost and time overruns can occur.

Design analysis significantly improves the accuracy of software size estimation. And, to ensure that adequate consideration is given to the design requirements, the developer, as a minimum, should be required to provide a software Work Breakdown Structure (WBS), a functional flow diagram, and estimates of software sizes for each work package prior to initiation of the development. If possible, algorithms to be programmed should also be provided by the developer; this is a method of ensuring that adequate thought has been given to the complexity of the problem.

5.1.3.1 Alternative approaches to software sizing. There are two methods used for sizing software early in the conceptual phase of development. The specific approach used is dictated by the experience of the estimator, and the extent and relevance of available historical data. Each of the approaches assume that the mission analysis is complete,

the operational requirements and major configuration constraints are identified, and that the operational concept, including its support concept, is understood. The methods entail the following:

- Specified computer hardware. The software sizing estimate is achieved by summing the core memory of the specified computer hardware, and adjusting the sum for assumed core overlays and expected core utilization.
- Software analogy. The software sizing estimate is derived by partitioning the software down to the functional sub-routine level, and then estimating the size of each work package by analogy with similar programs/subroutines/work packages in past development projects.

Sizing software by analogy, which requires the availability of relevant historical data, is considered the more accurate procedure. However, the success of the approach is highly dependent on the accuracy and relevancy of the data from which the analogy is being drawn. Consequently, software development activities are becoming increasingly aware of the need for historical data, and steps are being taken to assure its collection. If sufficient data exists, it is recommended that more than one approach be utilized and the results of the estimates be compared.

Software sizing in the conceptual phase is usually made in object code. This is because object code can usually be estimated early in the development with greater accuracy than source code. If both source and object code sizes are known, however, select source code because the error in the estimate would be less.

Work performed in the development of analytical models for estimating the size of software has been seriously constrained by the lack of relevant, detailed and accurate data. Often, relationships developed have included variables not considered valid determinants of size. High correlation between size and a variable is not justification for incorporating it in an estimating relationship.

Because of the importance of size to costing, an attempt was made to develop estimators to guide the sizing function. Variables considered as potential determinants of program size were selected from the SDC-Phase III data. Multivariate regression was performed using two sets of variables; word size and core size, essentially dictated by the processor used, were deleted as variables in one set. This permitted the development of a relationship which could be used without having selected the processor. Processor add time, also highly dependent on the processor selected, was not deleted because this variable can also be a specified requirement of the application. This analysis resulted in the following relationships:

$$\bullet \quad I = .449 \left[ \begin{array}{ccc} W_d & N_c & 0.360 \\ t_a & M_o & 0.105 \end{array} \right] \begin{array}{l} j=6 \\ \Pi f_j \\ j=1 \end{array} \quad (\text{Equation 7})$$

( $R^2 = .45$ , SE = 1.26 ln units)

<u>Factor</u>	<u>f</u>	<u>Yes</u>	<u>No</u>
Special display	$f_1 =$	0.550	1.000
On-line application	$f_2 =$	1.420	1.000
Software interfaces	$f_3 =$	1.453	1.000
Application involves more than one ADP center	$f_4 =$	1.947	1.000
Innovation required	$f_5 =$	2.271	1.000
Real-time application	$f_6 =$	6.913	1.000

$$\bullet \quad I = .017 \left[ \begin{array}{cccccc} W_d & N_c & t_a & C_s & W_s & 0.373 \\ & & & M_o & 0.145 & \end{array} \right] \begin{array}{l} j=6 \\ \Pi f_j \\ j=1 \end{array} \quad (\text{Equation 8})$$

( $R^2 = .45$ , SE = 1.26 ln units)

<u>Factor</u>	<u>f</u>	<u>Yes</u>	<u>No</u>
Special display	$f_1 =$	0.498	1.000
On-line application	$f_2 =$	1.316	1.000
Software interfaces	$f_3 =$	1.658	1.000
Application involves more than one ADP center	$f_4 =$	1.885	1.000
Innovation required	$f_5 =$	2.163	1.000
Real time application	$f_6 =$	7.217	1.000

where

$I$  = Program size, in thousands of lines of source code  
 $W_d$  = Size of data base, in thousands of words  
 $N_c$  = Number of classes of items in data base  
 $t_a$  = Add time of processor, in microseconds  
 $W_s$  = Size of memory word, in bits  
 $C_s$  = Core size, in thousands of words  
 $M_o$  = Number of message output types

The relative consistency in the values for the state variables,  $f_j$ , and in the exponents of the variables (except for  $t_a$ ) in the two relationships indicates that, except for the variable processor times, there appears to be little correlation between the variables core size and memory word sizes, and the other variables. The values for  $W_d$  and  $N_c$  are not likely to be available early in the software design, and as noted previously, the variables  $C_s$  and  $W_s$  are essentially dictated by the processor used. Also, the variables in the relationships are poor descriptors of the variance in size ( $R^2$  is low), and adding variables has no apparent effect on the  $R^2$  or the standard error. Therefore, the relationships are not considered satisfactory for use in sizing software, unless no other guidance is available.

5.1.3.2 Analysis of core requirements. A study sponsored by the Office of the Secretary of Defense, and performed jointly by the Johns Hopkins University Applied Physics Laboratory and the Mitre Corporation, [8], contained limited information relative to the memory requirements of software programs for several applications. Although this data contains deficiencies, the following relationship was derived for memory size using multivariate regression.

$$M = 0.177 k \left[ \frac{N_F^{0.337} W_s^{0.147}}{t_c^{0.770}} \right] \quad (\text{Equation 9})$$

$$(R^2 = .52, SE = .086 \text{ ln units})$$

where

M = Memory size in thousands of words of object code  
N<sub>F</sub> = Number of major functions to be performed by the software  
W<sub>S</sub> = Word size in bits  
t<sub>C</sub> = Cycle time of processor in microseconds  
k = A constant dependent on application =  
2.573 for signal processing  
2.727 for missile fire control  
2.781 for interfacing  
3.412 for communication  
3.565 for navigation  
4.046 for command and control  
4.451 for weapon fire control

Perhaps the variable N<sub>F</sub>, major function, is defined best by examples -- target tracking, target identification, navigation, system monitoring, display, steering, parameter measurement, tuning, target data entry, firing sequence control, etc. As in the previous relationships, t<sub>C</sub> and W<sub>S</sub> are essentially dictated by the CPU in the application. Based on the type applications, most of which are in real-time, few overlays would be expected. If, as in most cases, the core utilization can be assumed to be approximately 80 percent, the expression represents another estimator for software size.

5.1.3.2 Sizing by analogy. It is generally recommended that estimates of software size be derived through extrapolation from previously developed software. If the application is of the same type, e.g., command and control, and the functions of the system appear to be the same, direct projections with little adjustment for changes can be made. If the new application appears more complex (requires additional software functions), comparisons of the numbers and types of software functions will permit adjustments to be made to the words per function and to the number of functions.

In an analysis of software sizing, DAI has found that within software modules, the sizes of major functions of the module appear to decrease cumulatively at a nominal 80 percent slope. That is, if the number of functions

doubles, the average size per function decreases 20 percent. If the program size in object words, and the number of major functions performed by the software can be determined from analogous data, estimates can be made as to the size of new software as follows:

$$I_2 = \left[ \frac{I_1}{N_{f_1}} \right]^{.678} N_{f_2} \quad (\text{Equation 10})$$

where

- $I_2$  is the size of the proposed program in object words
- $I_1$  is the size of the analogous software in object words
- $N_{f_1}$  is the number of major functions performed by the analogous software
- $N_{f_2}$  is the projected number of major functions performed by the new software

5.1.4 Secondary resources. The main secondary resources are computer time, travel and documentation production. Based on the analysis of the SDC Phase III data, 4.91 computer hours are expended per man-month in batch processing. The coefficient of determination ( $R^2$ ) was .79 in a regression between these parameters. At \$60 per computer hour in a single partition batch environment, computer costs per man-month will approximate \$300. In multi-programming batch environments and interactive environments, where computer hours are not as meaningful, costs, however, will be approximately the same. At \$5000 per man-month for the primary resource, this secondary resource costs 6 percent of the primary resource. Five pages of deliverable documentation can be expected to be produced per man-month. At \$10 per page for production, this is estimated as 1 percent of the primary resource. Secondary resources such as travel can be considered to total up to about 5 percent of the primary resources, resulting in a total of 7.5 percent. Thus, the factor, k, in Equation 4 equals .075.

## 5.2 Proposed methodology and existing directives

The preceding paragraphs outlined a methodology for improved software cost estimating and recommended its application as proposed in Volume II of this Final Report. Concurrent with the development of the overall

approach, a review was conducted of Department of Defense and Air Force directives relative to software development to determine if major revisions were required to incorporate elements of the overall methodology. The review, which encompassed directives relative to techniques, methodology, guidance, definitions, data, etc., of system software acquisition, determined that the directives contain relevant information about which a software purchaser and developer should be cognizant. Unfortunately, for a new or relatively inexperienced software development manager the large number of applicable directives contain so much information to be assimilated that the effectiveness of the directives could be negated. Thus, attempting to revise the directives by injecting various aspects of the proposed overall software cost estimating methodology does not appear appropriate. Instead of reinforcing the proposed approach, the effect would be to obscure it.

What has been lacking in the field of software cost estimating has been a guidebook, which would combine techniques for improved software cost estimation with management considerations in a time-phased manner. Such a guide has been developed and is presented as Volume II of this report. Accordingly, revisions to software related directives are not proposed, but rather, it is recommended that application of the methodology contained in Volume II be endorsed for Air Force use and that it subsequently be revised iteratively as feedback is received from users. Further, it is suggested that a notice be issued, re-emphasizing that software project managers ensure that a conscientious effort be made to comply with appropriate existing directives such as for the development of a software Work Breakdown Structure (WBS) in accordance with MIL-STD-881.

### 5.3 Study results

Briefly, the study accomplished the following:

- Confirmed the need for a standardized set of definitions and for formalized mechanisms for collecting accurate software development data.
- Identified and quantified factors significantly affecting the accuracy of cost estimates.
- Proposed procedures for mitigating the adverse effects of these factors.
- Identified inadequacies existing in a frequently used tool of cost analysis, linear regression (conversely, established the benefits of non-linear regression).
- Derived a model for estimating the development time for software.
- Derived improved algorithms for estimating the manpower requirements of software development.
- Prescribed approaches for estimating the size of software.
- Integrated the control procedures and cost estimating algorithms into a guide for improved software cost estimation.

Over 100 factors identified as potential contributors to error in cost estimating were analyzed. Of these, 42 were found to have an impact, and 29 of the effects were quantified. Table 10 summarizes the impact of the factors on software development costs in terms of high, medium, low, and negligible levels of effect. Controls to mitigate the adverse impacts of the factors have been proposed. The effects of some factors can be profound. For example, standard errors in estimates of program size can attain 200 percent. Differences in definitions of instructions can cause errors of up to 300 percent in estimates. The magnitude of these errors make it obvious that the data collection and reporting associated with software development must be standardized before satisfactory accuracy in estimating development costs can be attained. The control mechanisms proposed by this study emphasize the need for such explicit delineation within the software development community.

A methodology for improved estimation of software development costs has been proposed. It has been integrated with the control mechanisms into a guidebook for estimating resource requirements and for improving the management of software development. The cost model (Equation 4) is a function of

TABLE 10. IMPACT OF FACTORS ON SOFTWARE COST ESTIMATION

Factor	Software application				
	Command and Control	Scientific	Business	Utility	All
Communication	H	H	H	H	H
Constrained, CPU Time	M	H	N	H	M
Constrained, Program Memory Size	M	M	M	L	M
Constrained, Time and Memory	H	H	H	H	H
Cost of Secondary Resources	N	N	N	N	N
Cost/Schedule Control Systems Criteria (C/SCSC)	N	N	N	N	N
Data Management Techniques	M	M	M	M	M
Data Collection, Amount and Method of	N	N	N	N	N
Developer's First Time on Specified Computer	H	H	H	H	H
Developer Using Another Activity's Computer	M	M	M	M	M
Development of Hardware, Concurrent	H	H	M	M	H
Development and Target Computer Different	H	L	M	N	M
Development Personnel Mix	M	M	M	M	M
Development Site	M	M	M	M	M
Development Site, Number of	M	M	M	M	M
Design Complexity	H	H	L	L	M
Design Stability	H	H	H	H	H
Instruction, Definition of	H	H	H	H	H
Innovation, Degree of	H	H	H	H	H
Programmer Testing	H	H	H	H	H

## LEGEND

H = High significant impact  
 M = Medium significant impact  
 L = Low significant impact  
 N = Negligible significant impact

TABLE 10. IMPACT OF FACTORS ON SOFTWARE COST ESTIMATION (Continued)

Factor	Software application				
	Command and Control	Scientific	Business	Utility	All
Programming Environment	H	H	H	H	H
Programming Facilities	H	H	H	H	H
Programming Techniques, Modern	H	H	H	H	H
Requirements, Language	H	H	H	H	H
Requirements, Maintainability	H	H	H	H	H
Requirements Changes, Operational	H	H	H	H	H
Requirements Definition, Operational	M	H	N	N	L
Requirements/Design Interface, Operational	H	H	H	H	H
Requirements, Quality	H	H	H	H	H
Requirements, Reliability	M	M	M	M	M
Requirements, Special Display	L	L	M	N	L
Requirements, Testing Including V&V	M	M	M	M	M
Requirements, Transportability	M	M	M	M	M
Requirements, User Considered	M	M	M	M	M
Sites, Multiple					
Software Utilization	M	M	M	M	M
Sizing Error	H	H	H	H	H
Software Development Schedule	H	H	H	H	H
Support Software Availability	H	H	H	H	H
Specified Response Time	M	H	H	N	M
Target CPU Designation	H	H	H	H	H
Work Breakdown Structure	H	H	H	H	H

## LEGEND

H = High significant impact  
 M = Medium significant impact  
 L = Low significant impact  
 N = Negligible significant impact

the manpower requirements (man-months), average cost of labor (dollars per man-month) and a factor reflecting the cost of secondary resources. This factor was determined to be 7.5 percent of the total labor cost. Since the average cost of labor is a user input to the model, the methodology emphasizes the estimation of manpower.

Manpower estimating relationships have been derived for all software, and for specific applications of software such as command and control, scientific, business and utility programs. Two types of estimating relationships have been developed: baseline, where the estimate is a function of software size, and multivariate, where expressions are in terms of software size and variables which reflect the impact of program parameters on the manpower requirements. The baseline relationships were derived using two analytical techniques, linear and non-linear regression. Non-linear regression was found to generate preferred estimators. The fit to the data and the statistical criteria (coefficient of determination and the standard error of the estimate) are significantly better. (Unfortunately, the algorithm available for this type of regression accepts only two variables. If it had accepted a greater number of variables, it could have been useful in enhancing the multivariate models developed with linear regression.) When possible, the estimating relationship for a specific application should be used. The estimates from the application models will be more accurate than those from the generalized models developed from the total population. Figure 50 reflects the suggested utilization of the estimating relationships.

The data used to support the study for the most part came from the SDC Phase III Study, [49]. This data base, considered the best assembled to date, has been analyzed several times. The approach in this study was quite different from prior analyses, and significant improvement in cost estimating models has been effected. However, inadequacies in the available data constrained the study results. To develop a better understanding of the effects of program parameters on software costs, accurate data at a more refined level of detail will have to be collected.

AD-A042 264

DOTY ASSOCIATES INC ROCKVILLE MD  
SOFTWARE COST ESTIMATION STUDY. VOLUME I. STUDY RESULTS.(U)  
JUN 77 J H HERD, J N POSTAK, W E RUSSELL

F/G 9/2

UNCLASSIFIED

TR-151

RADC-TR-77-220-VOL-1

F30602-76-C-0182

NL

3 of 4  
ADA042264



Application	Phases of Development	
	Concept Formulation Phase	Analysis and Design Phase
All Software - Object - Source	MM = 4.790 I <sup>0.991</sup> MM = 5.258 I <sup>1.047</sup>	MM = 4.790 I <sup>0.991</sup> MM = 5.258 I <sup>1.047</sup> (for I ≥ 10,000) MM = 5.258 I <sup>1.047</sup> MM = 2.060 I <sup>1.047</sup> (for I < 10,000) <sup>1</sup> MM = 2.060 I <sup>1.047</sup> ∏ <sub>j=1</sub> <sup>14</sup> f <sub>j</sub>
Command and Control - Object - Source	MM = 4.573 I <sup>1.228</sup> MM = 4.089 I <sup>1.263</sup>	MM = 4.573 I <sup>1.228</sup> MM = 4.089 I <sup>1.263</sup> (for I ≥ 10,000) MM = 4.089 I <sup>1.263</sup> MM = 0.501 I <sup>1.263</sup> (for I < 10,000) <sup>1</sup> MM = 0.501 I <sup>1.263</sup> ∏ <sub>j=1</sub> <sup>14</sup> f <sub>j</sub>
Scientific - Object - Source	MM = 4.495 I <sup>1.068</sup> MM = 7.054 I <sup>1.019</sup>	MM = 4.495 I <sup>1.068</sup> MM = 7.054 I <sup>1.019</sup> (for I ≥ 10,000) MM = 7.054 I <sup>1.019</sup> MM = 2.011 I <sup>1.019</sup> (for I < 10,000) <sup>1</sup> MM = 2.011 I <sup>1.019</sup> ∏ <sub>j=1</sub> <sup>14</sup> f <sub>j</sub>
Business - Object - Source	MM = 2.895 I <sup>0.784</sup> MM = 4.495 I <sup>0.781</sup>	MM = 2.895 I <sup>0.784</sup> MM = 4.495 I <sup>0.781</sup> (for I ≥ 10,000) MM = 4.95 I <sup>0.781</sup> MM = 3.742 I <sup>0.781</sup> (for I < 10,000) <sup>1</sup> MM = 3.742 I <sup>0.781</sup> ∏ <sub>j=1</sub> <sup>14</sup> f <sub>j</sub>
Utility - Object - Source	MM = 12.039 I <sup>0.719</sup> MM = 10.078 I <sup>0.811</sup>	MM = 12.039 I <sup>0.719</sup> MM = 10.078 I <sup>0.811</sup> (for I ≥ 10,000) MM = 10.078 I <sup>0.811</sup> MM = 1.744 I <sup>0.811</sup> (for I < 10,000) <sup>1</sup> MM = 1.744 I <sup>0.811</sup> ∏ <sub>j=1</sub> <sup>14</sup> f <sub>j</sub>

1. Can be used for budgeting for programs of I ≥ 10,000.

Figure 50. Suggested Utilization of Estimating Relationships for Development Manpower

Another serious limitation to the data base is that most of the programs are small (less than 10,000 instructions) and the largest data set is for 225,000 object (115,000 source) instructions. Therefore, estimates for development manpower are subject to significant errors associated with extrapolation outside the range of the data used in developing the estimating relationships.

General Research Corporation, in their study for ESD on software life-cycle costs, [59], proposed that economies of scale exist in software development, a phenomena contrary to almost all analysis results generated previously. This study has verified that economies of scale may exist, but in business type software development only. (A factor that could contribute to this apparent effect is that, although reported as newly written code, the developers of business programs may incorporate into their software recoverable code from other programs of a similar nature.)

Two methods for deriving estimates for software size have been described, (1) estimation by analogy, and (2) analysis of core requirements. Of the two methods, analogy is the preferred approach if data is available with which to develop the estimates. Once the computer has been selected, assumptions can be made relative to its memory utilization and concomitant program size considerations. As an adjunct to the core requirements method, a relationship has been developed for estimating memory requirements in terms of the number and type of major functions the software is to perform and characteristics of the memory data base. Other models for estimating software size were developed in which the independent variables are descriptors of the processor and the data base, information not usually known in software concept formulation. Neither the procedure nor the analytical models offer the accuracy desired for cost estimation. However, they can be used in lieu of more reliable guidance to software size.

To develop improved estimators for sizing software, it will be necessary that the collection of highly refined data be initiated. As an example, DAI is currently developing software sizing models for EW systems. To accomplish this, a historical data base is being developed by detailed reviews of program listings, hardware and software performance specifications, flow charts and several other data sources. This type of data collection ensures accurate and consistent data at necessary levels of detail.

A relationship has also been developed for estimating the time duration of software development. The algorithm, showing duration to be a function of the cube root of program size, can be used by software developers to assess the time that is required for completion of the software project.

Based on a preliminary examination of the factors that appear to affect duration, it seems likely that the relationship should include manloading as a variable. Further examination indicated that manloading may be in a complete form which would permit the determination of optimal duration (least cost) in terms of program size, manloading, and other relevant considerations.

#### 5.4 Recommendations

The study identified and quantified factors affecting the accuracy of software development cost estimates, and proposed procedures for controlling these factors. These were integrated, with algorithms defining the resource requirements of software development, into a methodology which has been presented as a guide for improved cost estimation. Following an evaluative review and validation, it is proposed that the guide be used to support the cost estimation process of the Air Force.

It should be considered a dynamic document. As data becomes available, the guide and its methodology should be updated to reflect factors not

able to be reflected in this study because of data limitations, and emerging technologies not addressed in the procedure.

As an extension of the work performed, consideration should be given the following:

- Data collection. Because continued improvement in software cost estimation is dependent upon the collection of accurate detailed software development data, it is recommended that a standardized set of definitions be developed and that mechanisms be implemented to collect the necessary data.
- Development duration. The duration of software development, especially for large programs, is usually dictated by an exogenous factor such as a hardware development schedule. Consequently, the developers of large software programs often do not have adequate time to complete their development properly. Thus, performance is traded off for development time. Based on preliminary examinations by DAI, duration more likely should be defined in terms of a complex relationship including program size and manpower loading among its variables. It also appears that the optimal (least cost) time duration may be able to be derived from this relationship. Consideration should be given to research for determining if such a relationship can be developed.
- Software sizing. In its present state, software sizing is inadequate to ensure accurate estimates for the size of projected software. Based upon the experience of DAI, it will require aggregation and collection of detailed data for a specific category of equipments, systems, or software to permit the derivation of relationships which express the size of software in terms of the functional and operational parameters of equipments (or systems). It is proposed that research into the development of such estimators for software sizing be undertaken.
- Automatic model update. Consideration should be given to the development and implementation of automated estimating models which are continually being updated as new data enters the data bank. Estimators, developed for each element of the software life cycle WBS, would be amended automatically with each entry of appropriate data. Agencies of DoD and the services have developed cost models of this type and have found them to be valuable guides to management.

- Non-linear regression. Non-linear regression used in deriving the baseline estimators resulted in much better fit to the data. And, for larger programs the fit appears to be as good as, if not better than, that obtained with linear multivariate models.

Most currently available multivariate regression packages utilize log transformations to handle non-linear regression models. This results in unequal weighting of the individual data points, with the heaviest weighting on the smaller values. As a consequence, coefficient determinations are often inaccurate. However, non-linear regression algorithms exist that enable direct fits to the data, thus avoiding this accuracy problem. Unfortunately, these non-linear programs are oriented to batch processing, and must be individually tailored for a particular application. The Marquardt algorithm has been designed to converge rapidly to a solution even from a distant starting point. There is a need to incorporate this algorithm for non-linear regression into a totally interactive system which will permit its application to the analysis of problems in a multivariate framework.

## APPENDIX A

### QUANTITATIVE DATA BASES OBTAINED FROM LITERATURE

#### Quantitative data bases suited for direct statistical analysis.

Described below in synoptic form are the ten data bases, presented in the literature, that were used to help support this study.

- |      |                          |  |
|------|--------------------------|--|
| A-1. | Reference Number:        | 46 (See Bibliography)  |
|      | Developing organization: | Systems Development Corporation (SDC)  |
|      | Year reported:           | 1964   |
|      | DAI designation          | SDC Phase I  |
|      | Number of observations:  | 27   |
|      | Number of variables:     | 98   |
|      | Description:             | Data from the initial study of an approximate seven year effort at SDC to study resource requirements for software development. All 27 observations were SDC developments. Resource requirements (dependent variables) were man-months, computer hours, and elapsed time. The sizing parameter was object instructions partitioned into both new and re-used code. |
| A-2  | Reference number:        | 132  |
|      | Developing organization: | SDC  |
|      | Year reported:           | 1965   |
|      | DAI designation:         | SDC Phase II   |
|      | Number of observations:  | 74   |
|      | Number of variables:     | 63   |
|      | Description:             | Data from second report from the SDC effort. All 74 observations were SDC developments, of which 23 are Phase I  |

data points. Resource requirements and sizing were reported in the same manner as in Phase I. There were 60 pure assembly level developments and 14 pure JOVIAL developments.

A-3    Reference number:            49  
       Developing organization:    SDC  
       Year reported:              1966  
       DAI designation:            SDC Phase III  
       Number of observations:    169  
       Number of variables:       94  
       Description:                Data from third report of the SDC effort. Of the 169 observations, 69 were retained from the SDC Phase II effort, and 101 were added from non-SDC organizations. Resource requirements were in the same terms as Phases I and II. Sizing was done in both object and source instructions, but was not partitioned into new and reused code. There were developments in both machine oriented languages (MOL) and a variety of high level languages (HOL) such as, FORTRAN, COBOL, and JOVIAL. There were also developments which were a mixture of both MOL and HOL. A wide variety of types of software were included: command and control, scientific, business, and support software.

A-4    Reference number:            120  
       Developing organization:    SDC  
       Year reported:              1967  
       DAI designation:            SDC Phase IV  
       Number of observations:    22  
       Number of variables:       146  
       Description:                All 22 developments were Spaceborne Software Systems containing information on on-board flight programs, ground programs, and support programs. Resource requirements were measured

in man-months, number of computer runs and elapsed time. Man-months and elapsed time were broken down into software life-cycle phases. Sizing was in object instructions with the amount of area reserved for data specified. The developments include the use of both MOL and HOL.

A-5. Reference number: 3  
Developing organization: Planning Research Corporation (PRC)  
Year reported: 1966  
DAI designation: PRC-1  
Number of observations: 18  
Number of variables: 84  
Description: Of the 18 developments, 16 fall into the category of general ADP, one was an embedded system for space tracking, and one was a command and control system. The 16 general ADP developments was a mixture of both business and scientific applications. Resource requirements were in man-months, computer hours, and elapsed time. In a number of cases both actual and estimated requirements were given. Sizing was given in both object and source instructions. Developments cover the use of both MOL and HOL languages.

A-6 Reference number: 104  
Developing organization: PRC  
Year reported: 1970  
DAI designation: PRC-2  
Number of observations: 20  
Number of variables: 93  
Description: This data base is often referred to as the ADPREP (Automatic Data Processing Resource Estimating Procedures) Data Base. The 20 systems were all Army systems in the general ADP business application area. Of the 20, 8 were

completely new developments and 12 were revisions of existing systems. Resource requirements were in man-months, computer hours, and elapsed time. Sizing was given in both object and source instructions. Developments cover the use of both MOL and HOL languages.

A-7. Reference number: 86  
Developing organization: International Business Machines Corporation (IBM)  
Year reported: 1975  
DAI designation: IBM-1  
Number of observations: 11  
Number of variables: 8  
Description: These were all IBM developments using structured top down programming. Most of them were embedded computer systems that were either SAMSO or SAMSO related, or other military or NASA systems. Resource requirements were in man-months and elapsed time. They include both support and operational software. Sizing is in source instructions. The developments cover the use of both MOL and HOL languages.

A-8 Reference number: Unpublished document  
Developing organization: IBM  
Year reported: 1975  
DAI designation: IBM-2  
Number of observations: 24  
Number of variables: 2  
Description: Like the data base just above, these were mainly software developments for computers embedded in weapons systems. It may include the above data base as a subset. The only variables given were resources in man-months and number of source instructions. Both the IBM data bases have a much smaller variance in productivity (instructions per man-months) than the other data bases.

A-9. Reference number: 110

Developing organization: Logicon, Incorporated

Year reported: 1968

DAI designation: Logicon

Number of observations: 7

Number of variables: 37

Description: These were all benchmark type problems (~500 source statements), each done by a single programmer in two different HOLs. The application areas were: two business, one interactive, one simulation, one scientific, and two data base management. The purpose was to compare PL/I, a very general language, against a competitive HOL language over a broad spectrum of application areas of possible future Air Force interest. Of the 37 variables, 25 relate to sizing, including both object and source, 2 to running time, 4 to resource expenditures in man-hours by software phase (analysis, design, code, and test), and 6 to machine resources in number of runs and total machine time for three different error categories. There is also much supplemental information about each of the developments, virtually all required information to do a thorough analysis except the actual computer listings.

A-10. Reference number: 81

Developing organization: Johns Hopkins University  
Applied Physics Laboratory (JHU/APL)

Year reported: 1975

DAI designation: JHU/APL

Number of observations: 39

Number of variables: 4

Description: This study, performed for OSD by JHU/APL and the MITRE Corporation, identified and defined software problems facing DOD, factors contributing to the problems, payoff areas and management policies needed to improve software management.

Included in the study was analyses of weapon system processing requirements. Data presented encompassed missile and weapon fire control, signal processing, command and control, and other applications. The types of information presented include memory requirements, functions performed, cycle time and word size.

#### BIBLIOGRAPHY

1. ASD Comptroller Automation Study, 24 March 1975.
2. Air Force Data Systems Design Center Manual, "Planning and Resource Management Information System," AFDSDCM 300-405, 1 September 1974.
3. Air Force, Hanscom Field, Bedford, Mass., "Air Force ADP Experience Handbook (Pilot Version)," ESD-TR-66-673, December 1966.
4. Air Force Systems Command Headquarters, Andrews Air Force Base, Md., "Laboratory Program Managers Guide--Part I, Statement of Work Preparation," June 1975.
5. Andres, Albert W., "An Executive Summary of a Study Report." Defense Systems Management School, Fort Belvoir, Virginia, May 1975.
6. Aron, Joel D., "Characteristics of Systems Development Life Cycle." IBM Corporation, 1973.
7. Aron, J.D., "Estimating Resources for Large Programming Systems." IBM Corporation.
8. Asch, A., et al., "DOD Weapon Systems Software Acquisition and Management Study, Volume I: Mitre Findings and Recommendations." MITRE Corp., MTR-6908, May 1975.
9. Asch, A., et al., "DOD Weapon Systems Software Acquisition and Management Study, Volume II: Supporting Material." MITRE Corp., MTR-6908, June 1975.
10. BMD Advanced Technology Center, "BMDATC Software Development System. Program Overview, Volume I." July 1975.
11. BMD Advanced Technology Center, "BMDATC Software Development System. Research Description, Volume II." July 1975.
12. Baker, F. Terry, and Mills, Harlan D., "Chief Programmer Teams." Datamation, December 1973, pp. 58-61.
13. Barry, Barbara S., and Naughton, John J., "Structured Programming Series. Volume X: Chief Programmer Team Operations Description." IBM Corporation, RADC-TR-74-300, January 1975, (A008861).
14. Bennett, Martha, et al., "Structured Programming Series. Volume VII. Addendum: Documentation Standards." IBM Corporation, RADC-TR-74-300, April 1975, (A016414).
15. Boehm, B.W., "Keynote Address: The High Cost of Software." TRW Systems Group, TRW-SS-73-08, September 1973.

16. Boehm, B.W., "Software and Its Impact: A Quantitative Assessment." TRW Systems Group, TRW-SS-73-04, May 1973.
17. Boehm, B.W., "Some Steps Toward Formal and Automated Aids to Software Requirements Analysis and Design." TRW Systems Group, TRW-SS-74-02, August 1974.
18. Boehm, Barry W., "Some Information Processing Implications of Air Force Space Missions: 1970-1980." Rand Corporation, RM-6213-PR, January 1970.
19. Boehm, Barry W., et al., "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980s." Space and Missile Systems Organization, SAMSO/XRS-71-1, February 1972.
20. Bourdon, Gerard A., "Software Cost Estimating--A Proposed Approach." December 29, 1975.
21. Brandon, Dick H., and Segelstein, Sidney, Esq., Data Processing Contracts. New York: Von Nostrand Reinhold Company, 1976.
22. Brooks, Frederick P., Jr., "The Mythical Man-Month." Datamation, December 1974, pp. 45-52.
23. Campbell, Donna G., et al., "Structured Programming Series. Volume III. ANS Cobol Precompiler Program Documentation." IBM Federal Systems Division, March 22, 1975.
24. Carter, Deane M., "A Study of Critical Factors in Management Information Systems for the U.S. Air Force." Colorado State University, Information Systems Series 460-2.
25. Cetron, M.J., "Technological Forecasting and the Computer Industry."
26. Cheatham, Thomas E., and Goldberg, Jack., "Proceedings of a Symposium on the High Cost of Software Held at the Naval Postgraduate School, Monterey, California, on September 17-19, 1973." Stanford Research Institute, AD-777-121, September 1973.
27. Civil Service Department, London: Her Majesty's Stationery Office, "Estimation, Planning and Control of Programming Activities." Central Computer Agency Guide No. 3, 1974.
28. Clark, R. Lawrence., "A Linguistic Contribution to GOTO-less Programming." Datamation, December 1973, pp. 62-62.
29. Connolly, J.T., "Software Acquisition Management Guidebook: Regulations, Specifications and Standards." The MITRE Corp., ESD-TR-75-91, October 1975.

30. Cooke, Lawrence H., Jr., "Programming Time vs. Running Time." Data-  
mation, December 1974, pp. 56-58.
31. Copes, W.S., "Conversion of Micom, Time-Phased, Life-Cycle Cost-  
Estimating... ." Army Materiel Systems Analysis Agency, AMSAA-TM-  
131, March 1972.
32. Defense Management Journal. October 1975.
33. Department of the Air Force, "Planning and Resource Management Infor-  
mation System: K055/PV." AFDSDCM 300-405, September 1, 1974.
34. Department of the Air Force, "Program Breakdown Structure and Codes."  
Air Force Systems Command Manual, AFSCM 173-4, November 24, 1972.
35. Department of the Air Force, the Army, and the Navy, "Cost/Schedule  
Control Systems Criteria Joint Implementation Guide." 31 March 1972.
36. Department of the Army, "Management Information Systems Handbook of  
ADP Resource Estimating Procedures (ADPREP)," DA Technical Bulletin  
TB 18-19-1, July 1975.
37. Departments of the Army, Navy, and the Air Force, "Final Report of  
the Joint Logistics Commanders Electronic Systems Reliability Work-  
shop." October 1, 1975.
38. Departments of the Army, the Navy, and the Air Force, "Joint Design-  
to-Cost Guide. A Conceptual Approach for Major Weapon System Acqui-  
sition." 3 October 1973.
39. Department of Defense, "Defense System Software Management Plan."  
March 1976.
40. Devenny, Thomas J., Capt. (THESIS), "An Exploratory Study of Soft-  
ware Cost Estimating at the Electronic Systems Division." Air  
Force Institute of Technology, Wright-Patterson AFB, Ohio, GSM/  
SM/76S-4, no date.
41. Dickson, K.G., "A Summary Description of the SDC Approach to Soft-  
ware Sizing and Cost Estimating." System Development Corporation,  
TM-5358/000/00, July 19, 1974.
42. Dinitto, Sam, "A Basic Methodology to Lower the High Cost of Soft-  
ware."
43. Dodson, E.N., et al., "Advanced Cost Estimating and Synthesis Tech-  
niques for Avionics. Final Report." General Research Corporation,  
Santa Barbara, California, CR-2-461, September 1975.
44. Donaldson, James R., "Structured Programming." Datamation, Decem-  
ber 1973, pp. 52-54.

45. Donelson, William S., "Project Planning and Control." Datamation, June 1976, pp. 73-80.
46. Farr, L., and Zagorski, H.J., "Factors that Affect the Cost of Computer Programming. Volume II: A Quantitative Analysis." System Development Corporation, Technical Documentary Report No. ESD-TDR-64-448, September 1964.
47. Farr, Leonard, et al., "Planning Guide for Computer Program Development." No. TM-2314/000/00A, System Development Corporation, 10 May 1965.
48. Fisher, David A., "Automatic Data Processing Costs in the Defense Department." Institute for Defense Analyses, Paper P-1046, October 1974.
49. Fleishman, T., "Current Results from the Analysis of Cost Data for Computer Programming." System Development Corporation, August 1966.
50. Fleiss, Joel E., et al., "A Statistics Gathering Package for the JOVIAL Language." Proprietary Software Systems, Inc., RADC-TR-73-381, January 1974, (775380).
51. Forman, E.H., and Singpurwalla, N.D., "An Empirical Stopping Rule for Debugging and Testing Computers." George Washington University, Serial T-320, August 1975.
52. Frederic, Brad C., "A Provisional Model for Estimating Computer Program Development Costs." Tecolote Research, Inc., TM-7/Rev. 1, December 1974.
53. Frost, Davis., "Designing for Generality." Datamation, December 1974, pp. 59-61.
54. Gates, Howard P., et al., "Electronics-X: A Study of Military Electronics with Particular Reference to Cost and Reliability. Volume 1: Executive Conspectus." Institute for Defense Analyses, January 1974.
55. Gates, Howard P., et al., "Electronics-X: A Study of Military Electronics with Particular Reference to Cost and Reliability. Volume 2: Complete Report." Institute for Defense Analyses, January 1974.
56. Gossick, Lee V., "Management of System Acquisition Programs in the Air Force." Defense Industry Bulletin, Summer 1971, pp. 23-29.
57. Gradwahl, Alan J., et al., "Phase II Final Report on Use of Air Force ADP Experience to Assist Air Force ADP Management, Volume I: Summary, Conclusions, and Recommendations." U.S. Air Force, Hanscom Field, Bedford, Massachusetts, ESD-TR-66-671, December 1966.

58. Gradwahl, Alan J., and Wootan, Welford O., Jr., "Phase II Final Report on Use of Air Force ADP Experience to Assist Air Force ADP Management. Volume III: Phase III Concept and Plan." U.S. Air Force, Hanscom Field, Bedford, Massachusetts, ESD-TR-66-671, December 1966.
59. Graver, C.A., et al., "Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software--Volume I: Study Results--Final Report." (Preliminary Draft), General Research Corporation, November 1976.
60. Hagan, S.R., and Knight, C.W., "An Air Force Guide for Monitoring and Reporting Software Development Status." The MITRE Corporation, ESD-TR-75-85, September 1975.
61. Halstead, M.H., "Language Selection for Applications." National Computer Conference, 1973, pp. 211-214.
62. Hanscom Air Force Base, Summary Notes of a Government/Industry Software Sizing and Costing Workshop held at Bedford, Massachusetts, October 1-2, 1974.
63. Hanscom Air Force Base, "Support of Air Force Automatic Data Processing Requirements Through the 1980s (SADPR-85)." ESD, 1974.
64. Hartwick, R. Dean (Logicon, Inc.), "The Advanced Targeting Study, Phase 1F, Vol. I, Ballistic Missile Software Technology Baseline," Space and Missile Systems Organization, Norton Air Force Base, California, Report No. SAMSO-TR-71-124 Vol. I, June 1971 (Reissued November 1971).
65. Helmer, F.T., et al., "Bibliography on Pricing Methodology and Cost Estimating." Air Force Academy, February 1972.
66. Honeywell, "Description and Use of the JOVIAL Programming Language in Both Batch and Time-Sharing Environments." Series 600/6000, No. BS06, Rev. 1, January 1973 (Honeywell Information Systems).
67. IBM, "Management Planning Guide for a Manual of Data Processing Standards." (GC20-1670-2).
68. Ikezawa, Michael A. (Logicon, Inc.), "The Advanced Targeting Study, Phase 1F, Vol. II. Ballistic Missile Computer Software Technology." Space and Missile Systems Organization, Norton Air Force Base, California, Report No. SAMSO-TR-71-124-Vol. II, July 1971 (Reissued November 1971).
69. James, L.E., et al., "Study of Reliability Prediction Techniques for Conceptual Phase." Hughes Aircraft Co., RADC-TR-74-235, (A001919).
70. JOCIT Compiler Users Manual, January 1974.

71. Johns Hopkins University, "DOD Weapon Systems Software Management Study. Appendix A, Findings and Recommendations of Previous Studies." APL/JHU SR 75-3A, June 1975.
72. Johns Hopkins University, "DOD Weapon Systems Software Management Study. Appendix C, Airborne Systems." APL/JHU SR 75-3C, June 1975.
73. Johns Hopkins University, "DOD Weapon Systems Software Management Study. Appendix D, Undersea and Landbased Systems." APL/JHU SR 75-3D, June 1975.
74. Johns Hopkins University, "DOD Weapon Systems Software Management Study. Appendix E, Bibliography." APL/JHU SR 75-3E, June 1975.
75. Kayfes, Richard E., et al. (Logicon, Inc.), "Advanced Targeting Study, Phase 1F, Volume V, Space Programming Language (Mark II) Compiler; Part B, User's Manual." Space and Missile Systems Organization, Norton Air Force Base, California, Report No. SAMSO-TR-71-124, Vol. V, August 1971.
76. Keider, Stephen P., "Why Projects Fail." Datamation, December 1974, pp. 53-55.
77. Kempes, Robert R., "Cost Performance Reporting and Baseline Management." Office of the Assistant Secretary of Defense (Comptroller).
78. Kessler, Marvin M., "Structured Programming Series. Volume XII: Training Materials." IBM Corporation, RADC-TR-74-300, Oct 1975, (A026947).
79. Kessler, Marvin M., and Kister, William E., "Structured Programming Series. Volume XIV: Software Tool Impact." IBM Corporation, RADC-TR-74-300, May 1975, (A015795).
80. Kessler, Marvin M., and Tinanoff, Nathan, "Structured Programming Series. Volume I: Programming Language Standards." IBM Corporation, RADC-TR-74-300, March 1975, (A016771).
81. Kossiakoff, A. et al., "DOD Weapon Systems Software Management Study." The Johns Hopkins University Applied Physics Laboratory, APL/JHU SR-75-3, June 1975.
82. Kosy, Donald W., "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology." Rand Corporation, R-1012-PR, June 1974.
83. Kraly, Thomas M., et al., "Structured Programming Series, Volume VIII: Program Design Study." IBM Corporation, RADC-TR-74-300, May 1975, (A016415).
84. Lipow, M., "Application of Algebraic Methods to Computer Program Analysis." TRW Systems Group, TRW-SS-73-10, May 1973.

85. Luppino, F.M., and Smith, R.L., "Structured Programming Series. Volume V: Programming Support Library Functional Requirements." IBM Corporation, RADC-TR-74-300, July 1974, (A003339).
86. Malone, John L., "Estimating Software Life Cycle Costs." IBM Corporation, January 1975.
87. Manley, John H., et al., "Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (SRWG Report). Volume I: Executive Summary." AD-A018-881, November 1975.
88. Manley, John H., et al., "Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (SRWG Report). Volume II: Supporting Technical Information." November 1975.
89. Manley, John H., et al., "Proceedings of the Aeronautical Systems Software Workshop." HQ AFSC/XRF, Andrews AFB, D.C., AFSC-TR-74-03, July 1, 1974.
90. McCracken, Faniel D., "Revolution in Programming--An Overview." Datamation, December 1973, pp. 50-52.
91. Miller, Edward F., Jr., and Lindamood, George E., "Structured Programming: Top-Down Approach." Datamation, December 1973, pp. 55-57.
92. MITRE Corporation, "An Air Force Guide to Contracting for Software Acquisition." ESD-TR-75-365, January 1976.
93. MITRE Corporation, "Engineering of Quality Software Systems." (8 volumes) RADC-TR-74-325, January 1975, Vol. I, (A007766), Vol. II (A007767), Vol. III, (A007768), Vol. IV, (A007769), Vol. V, (A007770), Vol. VI, (A007771), Vol. VII, (A007772), Vol. VIII, (A007773).
94. MITRE Corporation, "Project Guide to Content Requirements and Audience Needs." Project Staff (Project No. 572H). EDS-TR-75-308, December 1975.
95. Morin, Lois H., "Estimation of Resources for Computer Programming Projects." (Thesis) University of North Carolina, 1973.
96. Morris, J.M., et al., "Structured Programming Evaluation." Pattern Analysis & Recognition Corporation, RADC-TR-75-179, July 1975, (A015763).
97. Naughton, John J., and Smith, Ronald L., "Structured Programming Series. Volume XIII: Final Report." IBM Corporation, RADC-TR-74-300, July 1975, (A020858).
98. Naval Electronics Laboratory Center, "Suggestions for Designers of Navy Electronic Equipment 1975 Edition." AD-A014-296, 1975.

99. Navarro, Aaron B., et al., "Imbedded Software Monitors and Data Collection. Volume I: Design and Implementation." PRC Information Sciences Company, AD-787-672, September 1974.
100. Navarro, Aaron B., et al., "Imbedded Software Monitors and Data Collection. Volume II: System and Functional Program Description." PRC Information Sciences Company, September 1974.
101. Nelson, E.A., "Management Handbook for the Estimation of Computer Programming Costs." System Development Corporation, TM-3225/000/01, March 20, 1967.
102. Nelson, Eldred, "Developing a Software Cost Methodology." TRW Defense and Space Systems Group.
103. Ortega, L.H., "Structured Programming Series. Volume VII: Documentation Standards." IBM Corporation, RADC-TR-74-300, September 1974, (A008639).
104. Planning Research Corporation, "Automatic Data Processing Resource Estimating Procedures (ADPREP)." PRC R-1527, August 1970.
105. Premo, A.F., "Computer Software: Estimating Guidelines." Memorandum for the Record, Department of the Navy. ASW-117:AFP, 3360, Ser: 11/288, November 6, 1975.
106. Pryor, C. Nicholas, "A Comparative Description of Several High Level Computer Languages." Naval Surface Weapons Center, NSWC/WOL/TR 75-109, July 9, 1975.
107. Richards, P.K., et al., "Factors in Software Quality." General Electric Company Presentation under RADC Contract FO30602-76-C-0417, December 1976.
108. Rand Corporation. "Military Equipment Cost Analysis." June 1971.
109. Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques." TRW Systems Group, TRW-SS-70-01, August 1970.
110. Rubey, Raymond J., et al., "Comparative Evaluation of PL/I." Logicon, Inc., ESD-TR-68-150, April 1968.
111. Seward, Henry H., and Maust, Gregory E., "Automated Financial Analysis for Government Program Offices." Electronic Systems Division, Hanscom Field, Bedford, Massachusetts, ESD-TR-74-58, June 1974.
112. Shooman, Martin L., and Ruston, Henry, Polytechnic Institute of New York, "Summary of Technical Progress Software Modeling Studies." RADC-TR-76-143, Technical Report, May 1976, (A025895).

113. Smith, Ronald L., "Structured Programming Series. Volume IX: Management Data Collection & Reporting." IBM Corporation, RADC-TR-74-300, October 1974, (A008640).
114. Smith, Ronald L., "Structured Programming Series. Volume XI: Estimating Software Project Resource Requirements." IBM Corporation, RADC-TR-74-300, January 1975, (A016416).
115. Smith, Ronald L., "Structured Programming Series. Volume XV: Validation & Verification Study." IBM Corporation, RADC-TR-74-300, May 1975, (A016668).
116. Softech, Inc., "Support Software Planning Study." Final Report for NADC, March 20, 1974.
117. Software Management Conference, "Abridged Proceedings," First Series, 1976.
118. Space and Missile Systems Organization, "User's Manual for the Logistics Support Cost Model Computer Program." Version 1.3, February 1976.
119. Sukert, Alan N., Capt., "A Software Reliability Modeling Study." RADC-TR-76-247, In-House Report, August 1976, (A030437).
120. System Development Corporation, "Spaceborne Software Systems Study. Volume I: Summary." Report No. SSD-TR-67-11, January 1967.
121. System Development Corporation, "Spaceborne Software Systems Study. Volume II: Survey, Analysis and Recommendations." Report No. SSD-TR-67-11, January 1967.
122. System Development Corporation, "Spaceborne Software Systems Study. Volume III: Recommendation for a Common Space Programming Language." Report No. SSD-TR-67-11, January 1967.
123. System Development Corporation, "Spaceborne Software Systems Study. Volume IV: Survey of Software Techniques." Report No. SSD-TR-67-11, January 1967.
124. TRW Systems Group, "Software Reliability Study." Report No. 74-2260-1.9-29, October 1974.
125. TRW Systems Group, "Software Development Characteristics Study for the CCIP-85 Study Group. Vol. B (Appendices). Final Report." No. 4851.1-002, 8 October 1971.
126. TRW Defense & Space Systems Group, "Software Reliability Study." RADC-TR-76-238, Final Technical Report, August 1976, (A030798).

127. TRW Systems Group, "TRW Software Series: Index to Publications in Print." TRW-SS-INDEX Issue No. 3, January 1976.
128. Tinanoff, Nathan, "Structured Programming Series. Volume II: Pre-Compiler Specifications." IBM Corporation, RADC-TR-74-300, May 1975, (A018046).
129. Tinanoff, Nathan, and Luppino, Fred M., "Structured Programming Series. Volume VI: Programming Support Library Program Specifications." IBM Corporation, RADC-TR-74-300, February 1975, (A007796).
130. Trainor, W.Lynn, et al., "Software Design and Verification System (SDVS)." TRW Systems Group, August 1973.
131. Trimble, John T., "Structured Programming Series. Volume IV: Data Structuring Study." IBM Corporation, RADC-TR-74-300, April 1975, (A015794).
132. Weiwurm, G.F., et al., "Research Into the Management of Computer Programming: A Transitional Analysis of Cost Estimation Techniques." System Development Corporation, TM-2712/000/00, November 12, 1965.
133. Willmorth, N.E., "Software Development Data Collection Survey of Project Managers." System Development Corporation, TM-5542/005/00, February 20, 1976.
134. Willmorth, N.E., and Finfer, M.C., "Appendix A--Compendium of Data Parameters for Productivity and Reliability Studies." System Development Corporation, TM-5542/007/00.
135. Wolverton, R.W., "Paradoxes in Management: Software Standards and Procedures." TRW Systems Group, TRW-SS-74-11, April 1974.
136. Wolverton, R.W., "The Cost of Developing Large Scale Software." TRW Systems Group, TRW-SS-72-01, March 1972.
137. Wolverton, R.W., and Schick, G.J., "Assessment of Software Reliability." TRW Systems Group, TRW-SS-72-04, September 1972.

*MISSION  
of  
Rome Air Development Center*

*RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C<sup>3</sup>) activities, and in the C<sup>3</sup> areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.*



AD-A042 264

DOTY ASSOCIATES INC ROCKVILLE MD

F/G 9/2

SOFTWARE COST ESTIMATION STUDY. VOLUME I. STUDY RESULTS.(U)

JUN 77 J H HERD, J N POSTAK, W E RUSSELL

F30602-76-C-0182

UNCLASSIFIED

TR-151-VOL-1

RADC-TR-77-220-VOL-1

NL

4 OF 4  
ADA  
042264

SUPPLEMENTARY

INFORMATION

END  
DATE  
FILMED

6-79  
DDC

**SUPPLEMENTARY**

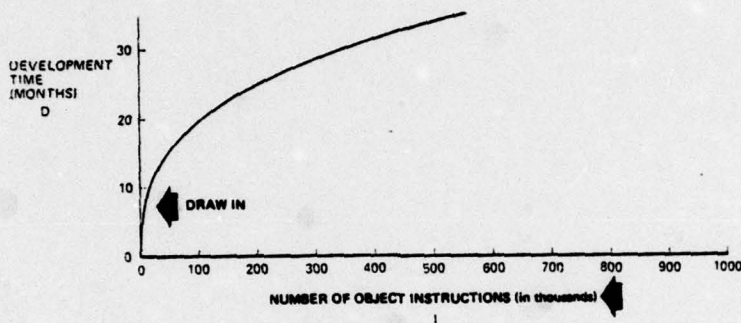
**INFORMATION**

# Doty Associates, Inc.

## Errata Sheet for Software Cost Estimation Study, Study Results, Final Technical Report, Volume I, June 1977, RADC-TR-77-220 (NTIS No. AD-A042264)

Corrections should be made to the above referenced volume as follows:

- Page 44. Figure 5. Draw in section of curve and add "(in thousands)" after "Number of Object Instructions", as noted below:



- Page 113. Paragraph 4.3.41. Second Line. Change "15 percent" to read "7.5 percent".
- Page 117. Paragraph 5.1.1. Equation 3 and definitions. Change to read as follows:

$$D = \frac{1000 I}{a + bI^{.667}}$$

where

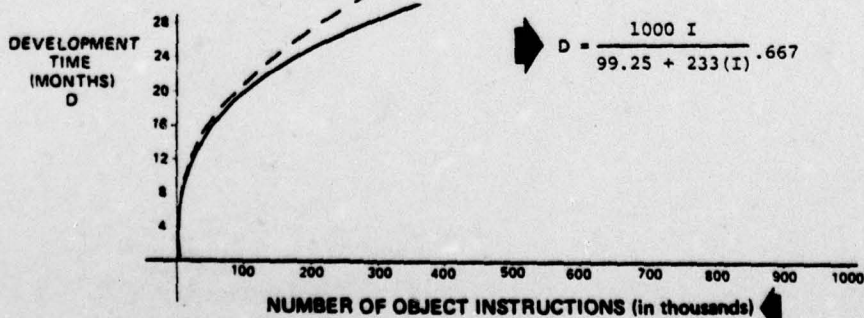
D is the duration of development, in months

I is the size of the program, in thousands of words of object code

a is a constant (= 99.25)

b is a constant (= 233)

- Page 119. Figure 8. Add "(in thousands)" after "Number of Object Instructions" and revise the equation as noted below:



$$D = \frac{1000 I}{99.25 + 233(I)^{.667}}$$

# Doty Associates, Inc.

- Page 173. Equation 9. Revise using the below attached self-adhesive overlay.

$$M = e^{0.177+k} \left[ \frac{N_F^{0.337} W_S^{0.147}}{t_c^{0.770}} \right]$$

- Page 181. Figure 50. Add "Subsequent Phases" to large arrow at top of figure. Change four commas to decimal points where indicated. In Business Application, change equation "MM = 4.951<sup>0.781</sup>" to read "MM = 4.4951<sup>0.781</sup>".

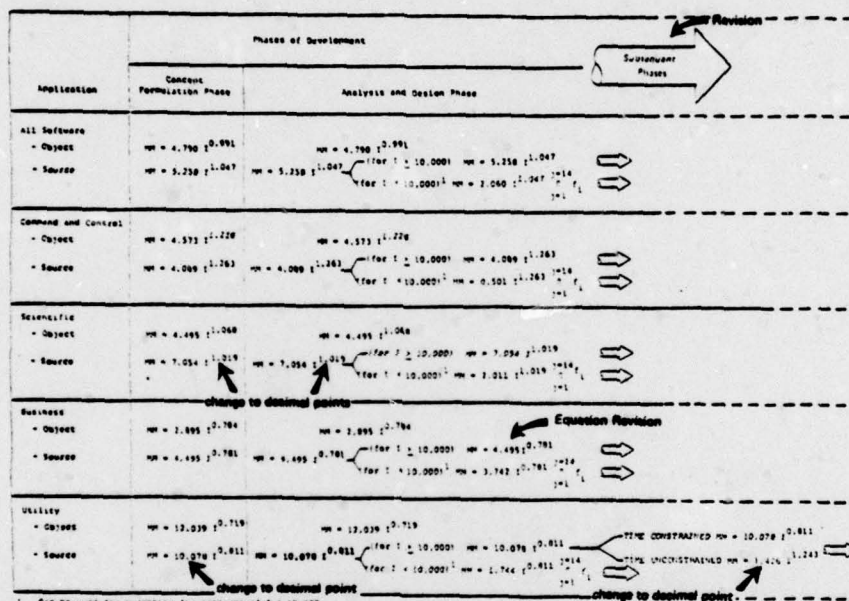


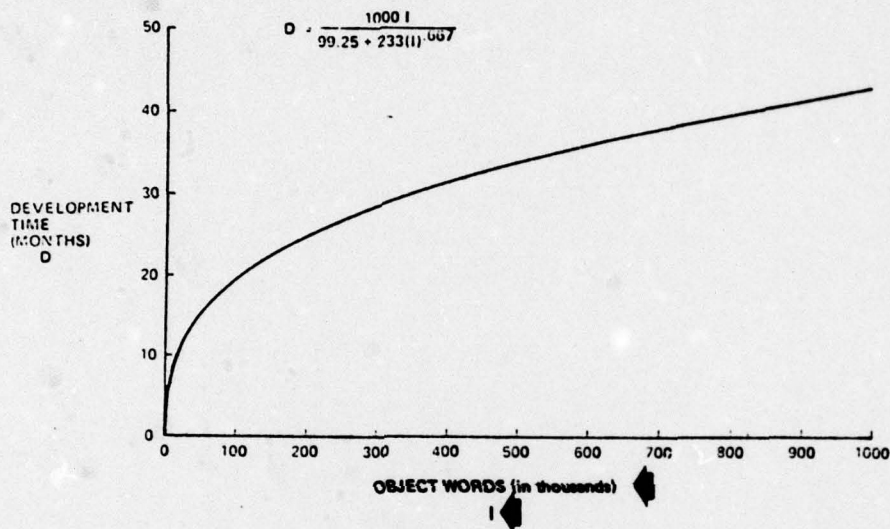
Figure 50. Suggested Utilization of Estimating Relationships for Development Manpower

# Doty Associates, Inc.

## Errata Sheet for Software Cost Estimation Study, Guidelines for Improved Software Cost Estimating, Final Technical Report, Volume II, August 1977, RADC-TR-77-220 (NTIS No. AD-A044609)

Corrections should be made to the above referenced volume as follows:

- Page 17. Figure 4. After "Object Words", delete "(000)" and add "(in thousands)". Also add "I" below "Object Words".



- Page 17. Equation (7). Revise definition of "I" to read "Number of thousands of delivered object instructions."
- Page C-37. Revise figure as indicated for Figure 4 on page 17 above.
- Page D-6. Paragraph D.2.1. Revise the equation by using the below attached self-adhesive overlay.

$$M = e^{0.177+k} \left[ \frac{N_F^{0.337} W_S^{0.147}}{t_C^{0.770}} \right]$$